University of California
Santa Barbara

Regularization and Look-Ahead Procedures for Selection of Basis Functions from Multiple Libraries

A dissertation submitted in partial satisfaction
of the requirements for the degree

Doctor of Philosophy
in
Statistics

by

Ling Zhu

Committee in charge:

Professor Yuedong Wang, Chair
Professor Wendy Meiring
Professor John Hsu

March 2017

The Dissertation of Ling Zhu is approved.

Professor Wendy Meiring

Professor John Hsu

Professor Yuedong Wang, Committee Chair

March 2017

Regularization and Look-Ahead Procedures for Selection of Basis Functions from Multiple Libraries

by

Ling Zhu

I want to dedicate this work to my parents,
for their unwavering love and encouragement.

Acknowledgements

First and foremost, I would like to express my most sincere gratitude to my advisor, Professor Yuedong Wang. His dedication to teaching and statistical research has constantly inspired me during my graduate study. I am very grateful for all the knowledge I gained from attending the excellent classes he has offered. His patient guidance and valuable insight have helped me conquer numerous obstacles in my research and made the work presented in dissertation possible. I would also like to thank my committee members, Professor Wendy Meiring and Professor John Hsu, who gave me great encouragement and advice. I am indebted to Professor Meiring for her tremendous help in improving the accuracy of my writing, as well as her expert opinions regarding the real application of my methodology.

My deep appreciation also goes to all other faculty members, staff, and colleagues in the Statistics department. Together you have created a nourishing and friendly academic environment that helped me grow. I will cherish every moment I spent in the research seminars and every conversation I had with my colleagues regarding research and life. We always had support for each other and thanks to you my time at UC Santa Barbara was absolutely wonderful.

Finally, I want to thank my parents, Jianxun Zhu and Hongmei Cai, for their unconditional love and support. Their unwavering encouragement gave me strength and confidence. I owe everything to them.

Curriculum Vitæ

Ling Zhu

Education

2016	Doctor of Philosophy in Statistics and Applied Probability, Department of Statistics and Applied Probability, University of California, Santa Barbara.
2010	Master of Arts in Economics, Department of Economics, University of California, Santa Barbara.
2009	Bachelor of Arts in Economics, Zhejiang Gongshang University, Hangzhou, China.

Education

2012-2016	Teaching Assistant, Department of Statistics and Applied Probability, University of California, Santa Barbara.
2013	Research Intern, EdLab, Teachers College, Columbia University, New York, New York.
2011	Readership Assistant, Department of Statistics and Applied Probability, University of California, Santa Barbara.

Abstract

Regularization and Look-Ahead Procedures for Selection of Basis Functions from Multiple Libraries

by

Ling Zhu

Basis Selection from Multiple Libraries (BSML) procedure was proposed by Sklar et al. (2013) to estimate spatially inhomogeneous functions with linear combinations of basis functions that are adaptively selected from multiple libraries. This methodology proves to be successful but suffers from certain drawbacks. First, the BSML procedures utilize the generalized degrees of freedom (GDF) to measure the complexity of a modeling procedure. This approach can be computationally demanding, since the GDF needs to be estimated at every step of the forward selection process. Another drawback of the BSML procedures is its greedy nature when searching for basis functions. At each forward selection step, the BSML procedures search for only one basis function to add to the current model.

We first propose two procedures called Adaptive LASSO Basis Selection (ALBS) and its modified version ALBS-2 to address the first drawback. The idea is to penalize the bases differently based on their induced reductions of RSS and the libraries they are from. This regularization approach shows no clear advantages over the BSML procedures in terms of performance according to our simulations.

We then propose the look-ahead procedure (LAP) to address both drawbacks of the BSML procedures. LAP avoids the estimation of GDF by treating the inflated degrees of freedom (IDF) of each library as tuning parameters, and then estimate them using cross validation. Moreover, LAP adopts less greedy search rules in the forward selection

so that either one or a pair of basis functions can be added to the current model after each iteration. Extensive simulations show that the LAP can outperform the BSML procedures and other widely-used modeling procedures both in terms of computation speed and mean squared error. Interesting real data applications in geology and meteorology using the look-ahead procedure are also provided.

Contents

Chapter 1

Introduction

1.1 Nonparametric Regression with Splines

Consider the following univariate nonparametric regression problem:

$$y_i = f(x_i) + \varepsilon_i, \quad i = 1, \ldots, n, \tag{1.1}$$

where y_i and x_i are the ith observation of the response variable Y and independent variable X respectively, ε_i's are random errors with mean zero and variance σ^2, and f is the regression function to be estimated on the domain $\mathcal{X} = [0, 1]$. We do not assume f to have a predetermined form. Instead, we only require f to have certain qualitative properties, such as smoothness, and let data decide the form of f. Thus, the model considered in (1.1) is nonparametric. There are two basic spline methods that can be used to estimate f: regression spline and smoothing spline.

1.1.1 Regression Spline

The regression spline model assumes that f can by well-approximated by either polynomial spline, or natural polynomial spline. Let $\pi^r = \{f : f(x) = \beta_0 + \beta_1 x + \beta_2 x^2 + \cdots + \beta_{r-1} x^{r-1}\}$ be the collection of polynomials of order r, and C^r be the set of functions such that their r-th derivatives are continuous at each x. Then a polynomial spline f of order r with interior knots $\boldsymbol{t} = (t_1, \ldots, t_k)$, where $0 = t_0 < t_1 < \cdots < t_k < t_{k+1} = 1$, satisfies:

(i) $f(x) \in \pi^r$, for $x \in [t_i, t_{i+1}]$, $i = 0, \ldots, k$.

(ii) $f \in C^{r-2}$, and its $(r-1)$-st derivative is a step function with jumps at the knots.

We denote the collection of all such polynomial splines of order r with interior knots $\boldsymbol{t}$ as $S^r(\boldsymbol{t})$. For f to be a natural polynomial spline, it has to satisfy an additional condition:

(iii) When $r = 2m$, $f(x) \in \pi^m$ for $x \in [0, t_1]$ and $x \in [t_k, 1]$.

This translates to the boundary conditions on the natural splines: $f^{(j)}(0) = f^{(j)}(1) = 0$, for $j = m, \ldots, 2m-1$, where $f^{(j)}$ is the jth derivative of f. We denote the collection of all such natural splines of order $2m$ with interior knots $\boldsymbol{t}$ as $NS^{2m}(\boldsymbol{t})$.

The space $S^r(\boldsymbol{t})$ is spanned by $r+k$ basis functions: $\{1, x, \cdots, x^{r-1}, (x-t_1)_+^{r-1}, \cdots, (x-t_k)_+^{r-1}\}$, so for any $f \in \mathcal{S}^r(\boldsymbol{t})$, it can be uniquely represented by

$$f(x) = \sum_{j=0}^{r-1} \theta_j x^j + \sum_{j=1}^{k} \delta_j (x - t_j)_+^{r-1}. \tag{1.2}$$

Let $\boldsymbol{X_t}$ be the $n \times (r+k)$ design matrix with the basis functions of $\mathcal{S}^r(\boldsymbol{t})$ as its columns, where the ith row corresponds to the $r+k$ basis functions evaluated at observation location x_i for $i = 1, \ldots, n$. Then (1.1) can be written as:

$$\boldsymbol{y} = \boldsymbol{X_t}\,\boldsymbol{\beta_t} + \boldsymbol{\varepsilon}, \tag{1.3}$$

where $\boldsymbol{y} = (y_1, \ldots, y_n)'$, $\boldsymbol{\varepsilon} = (\varepsilon_1, \ldots, \varepsilon_n)'$, and $\boldsymbol{\beta_t} = (\theta_0, \ldots, \theta_{r-1}, \delta_1, \ldots, \delta_k)'$. The estimate of $\boldsymbol{\beta_t}$ that minimizes the residual sum of squares $\mathrm{RSS}(\boldsymbol{t}) = (\boldsymbol{y} - \boldsymbol{X_t}\,\boldsymbol{\beta_t})'(\boldsymbol{y} - \boldsymbol{X_t}\,\boldsymbol{\beta_t})$ is:

$$\hat{\boldsymbol{\beta}}_{\boldsymbol{t}} = (\boldsymbol{X}_{\boldsymbol{t}}'\,\boldsymbol{X}_{\boldsymbol{t}})^{-1}\boldsymbol{X}_{\boldsymbol{t}}'\,\boldsymbol{y}. \tag{1.4}$$

The solution in (1.4) is for fixed knots $\boldsymbol{t} = (t_1, \ldots, t_k)$. However, both the number and the locations of the knots in $\boldsymbol{t}$ should be treated as tuning parameters. Some ad hoc rules are available for selecting $\boldsymbol{t}$ adaptively (See Eubank (1999)). The generalized cross validation (GCV) criterion can also be used to select both the knot locations and number of interior knots (k):

$$\mathrm{GCV}(\boldsymbol{t}) = \frac{\mathrm{RSS}(\boldsymbol{t})}{\big(n - (r+k)\big)^2}, \tag{1.5}$$

where $r + k$ is the total number of basis functions. The tuning parameter $\hat{\boldsymbol{t}}$ is selected to minimize the GCV in (1.5).

The scenario is similar when we assume $f \in NS^{2m}(\boldsymbol{t})$. It is obvious that $NS^{2m}(\boldsymbol{t}) \subset S^{2m}(\boldsymbol{t})$ since a natural spline has to satisfy the boundary condition (iii) as well. Because there are a total of $2m$ such boundary constraints, the dimension of $NS^{2m}(\boldsymbol{t})$ is $r + k - 2m = k$, i.e., $NS^{2m}(\boldsymbol{t})$ is spanned by only k basis functions. Any $f \in NS^{2m}(\boldsymbol{t})$ can be represented by

$$f(x) = \sum_{j=0}^{m-1} \theta_j x^j + \sum_{j=1}^{k} \delta_j (x - t_j)_+^{2m-1}. \tag{1.6}$$

1.1.2 Smoothing Spline

The smoothing spline model assumes f to be in certain functional space, such as the Sobolev space $W_2^m[0,1]$:

$$W_2^m[0,1] = \left\{ f : f, f^{(1)}, \cdots, f^{(m-1)} \text{ are absolutely continuous }, \int_0^1 \big(f^{(m)}(x)\big)^2 \mathrm{d}x < \infty \right\}. \tag{1.7}$$

Then f is estimated by solving the following penalized least square (PLS) problem:

$$\hat{f}_\lambda = \underset{f \in W_2^m[0,1]}{\operatorname{argmin}} \frac{1}{n} \sum_{i=1}^{n} \big(y_i - f(x_i)\big)^2 + \lambda \int_0^1 \big(f^{(m)}(x)\big)^2 \mathrm{d}x, \tag{1.8}$$

where λ is the tuning parameter that balances the trade-off between the goodness of fit and the smoothness of $\hat{f}_\lambda$. Assuming the design points are distinct, then for a fixed λ, the solution $\hat{f}_\lambda$ belongs to $NS^{2m}(x_1, \ldots, x_n)$ (See Eubank (1999)). When $\lambda = 0$, $\hat{f}_\lambda$ is a natural spline that interpolates the data, since there is no penalty on the roughness of $\hat{f}_\lambda$. On the other hand, when $\lambda = \infty$, $\hat{f}_\lambda$ becomes an m-th order polynomial. A commonly used criterion for selecting the smoothing parameter λ is the GCV (See Craven and Wahba (1978)). Let $\hat{\boldsymbol{f}}_\lambda = \big(\hat{f}_\lambda(x_1), \ldots, \hat{f}_\lambda(x_n)\big)'$, and $\boldsymbol{P}_\lambda$ be the hat matrix that satisfies $\hat{\boldsymbol{f}}_\lambda = \boldsymbol{P}_\lambda \boldsymbol{y}$, then the GCV criterion is defined as

$$\mathrm{GCV}(\lambda) = \frac{\frac{1}{n} \sum_{i=1}^{n} \big(y_i - \hat{f}_\lambda(x_i)\big)^2}{\big[\frac{1}{n} \mathrm{tr}\big(\boldsymbol{I} - \boldsymbol{P}_\lambda\big)\big]^2}, \tag{1.9}$$

where $\boldsymbol{I}$ is the $n \times n$ identity matrix, and $\mathrm{tr}(\cdot)$ returns the trace of a matrix. The GCV estimate of λ is the minimizer of $\mathrm{GCV}(\lambda)$. Other criteria such as the unbiased risk (UBR) criterion and generalized maximum likelihood (GML) criterion can also be used to select λ (see Wahba (1990)).

The Sobolev space $W_2^m[0,1]$ is an example of a reproducing kernel Hilbert space (RKHS), where a reproducing kernel (RK) $R(x, \cdot)$ is defined. More generally, let $\mathcal{H}$ be a RKHS with the domain $\mathcal{X}$, where $\mathcal{X}$ is an arbitrary set, then $R(x, \cdot)$ is in $\mathcal{H}$ with the reproducing property:

$$\langle R(x, \cdot), f \rangle = f(x) \tag{1.10}$$

for any $x \in \mathcal{X}$ and $f \in \mathcal{H}$, where $\langle \cdot, \cdot \rangle$ is the inner product on $\mathcal{H}$. $R(x, \cdot)$ is symmetric and non-negative definite. Meanwhile, it has other nice properties. For example, given

the tensor sum deposition $\mathcal{H} = \mathcal{H}_0 \oplus \mathcal{H}_1$, we have $R = R_0 + R_1$ where R, R_0, and R_1 are the RK's of $\mathcal{H}, \mathcal{H}_0$, and $\mathcal{H}_1$ respectively.

We take the Sobolev space $W_2^m[0,1]$ as an example (Wang (2011)). Define the reproducing kernel as $R_m(x,z) = \sum_{\nu=0}^{m} k_\nu(x)k_\nu(z) + (-1)^{m-1}k_{2m}(|x-z|)$ for $x, z \in [0,1]$, where $k_r(x) = B_r(x)/r!$ are the scaled rth Bernoulli polynomials, with $B_0(x) = 1$, $B_r'(x) = rB_{r-1}(x)$, and $\int_0^1 B_r(x)\mathrm{d}x = 0$ for $r = 0, 1, 2, \ldots$. The first five scaled Bernoulli polynomials are given below.

$$
\begin{aligned}
k_0(x) &= 1, \\
k_1(x) &= x - 0.5, \\
k_2(x) &= \frac{1}{2}\left[k_1^2(x) - \frac{1}{12}\right], \\
k_3(x) &= \frac{1}{6}\left[k_1^3(x) - \frac{1}{4}k_1(x)\right], \\
k_4(x) &= \frac{1}{24}\left[k_1^4(x) - \frac{1}{2}k_1^2(x) + \frac{7}{240}\right].
\end{aligned} \tag{1.11}
$$

Then for the tensor sum decomposition $W_2^m[0,1] = \mathcal{H}_0 \oplus \mathcal{H}_1$, where

$$\mathcal{H}_0 = \mathrm{span}\big\{k_0(x), k_1(x), \ldots, k_{m-1}(x)\big\}, \tag{1.12}$$

$$\mathcal{H}_1 = \left\{f : \int_0^1 f^{(\nu)}(x)\mathrm{d}x = 0,\ \nu = 0, \ldots, m-1, \quad \int_0^1 \big(f^{(m)}(x)\big)^2\mathrm{d}x < \infty\right\}, \tag{1.13}$$

their corresponding RKs are:

$$R_0(x,z) = \sum_{\nu=0}^{m-1} k_\nu(x)k_\nu(z), \tag{1.14}$$

$$R_1(x,z) = k_m(x)k_m(z) + (-1)^{m-1}k_{2m}(|x-z|), \tag{1.15}$$

under the corresponding inner products:

$$\langle f, g\rangle_0 = \sum_{\nu=0}^{m-1} \int_0^1 f^{(\nu)}(x)\mathrm{d}x \int_0^1 g^{(\nu)}(x)\mathrm{d}x, \tag{1.16}$$

$$\langle f, g\rangle_1 = \int_0^1 f^{(m)}(x) g^{(m)}(x)\mathrm{d}x. \tag{1.17}$$

Thus, the penalty term $\int_0^1 \left(f^{(m)}(x)\right)^2 \mathrm{d}x$ in the objective function (1.8) is the same as $\|P_1 f\|^2$, where P_1 is the projection of $W_2^m[0,1]$ onto $\mathcal{H}_1$. Let $\{\phi_1(x), \ldots, \phi_m(x)\}$ be the basis of $\mathcal{H}_0$, where $\phi_\nu(x) = k_{\nu-1}(x)$, for $\nu = 1, \ldots, m$. Also, define the representers as $\xi_i(x) = R_1(x, x_i)$ for $i = 1, \ldots, n$. Then the Kimeldorf–Wahba representer theorem states that the minimizer $\hat{f}_\lambda$ in (1.8) has the representation

$$\hat{f}_\lambda(x) = \sum_{\nu=1}^{m} d_\nu \phi_\nu(x) + \sum_{i=1}^{n} c_i \xi_i(x). \tag{1.18}$$

Based on (1.18), the solution in (1.8) can be written as

$$\hat{\boldsymbol{f}}_\lambda = \boldsymbol{T}\boldsymbol{d} + \boldsymbol{\Sigma}\boldsymbol{c}, \tag{1.19}$$

where $\boldsymbol{T} = \left\{\phi_\nu(x_i)\right\}_{i=1\,\nu=1}^{n\quad m}$, $\boldsymbol{\Sigma} = \left\{R_1(x_i, x_j)\right\}_{i,j=1}^{n}$, $\boldsymbol{d} = (d_1, \ldots, d_m)'$, $\boldsymbol{c} = (c_1, \ldots, c_n)'$, $\hat{\boldsymbol{f}}_\lambda = \left(\hat{f}_\lambda(x_1), \ldots, \hat{f}_\lambda(x_n)\right)'$. Thus, the minimization problem in (1.8) is equivalent to:

$$\underset{\boldsymbol{c},\boldsymbol{d}}{\operatorname{argmin}} \frac{1}{n}\|\boldsymbol{y} - (\boldsymbol{T}\boldsymbol{d} + \boldsymbol{\Sigma}\boldsymbol{c})\|^2 + \lambda \boldsymbol{c}'\boldsymbol{\Sigma}\boldsymbol{c}. \tag{1.20}$$

The coefficient vectors $\boldsymbol{d}$ and $\boldsymbol{c}$ in (1.20) can be determined by solving the following

equations:

$$\begin{aligned}(\boldsymbol{\Sigma} + n\lambda \boldsymbol{I})\boldsymbol{c} + \boldsymbol{T}\boldsymbol{d} &= \boldsymbol{y}, \\ \boldsymbol{T}'\boldsymbol{c} &= \boldsymbol{0}.\end{aligned} \tag{1.21}$$

1.2 Smoothing Spline ANOVA

Now consider the multivariate nonparametric regression problem:

$$y_i = f(\boldsymbol{x}_i) + \varepsilon_i, \quad i = 1, \ldots, n, \tag{1.22}$$

where $\boldsymbol{x}_i = (x_{i1}, \ldots, x_{id})$ is the vector of values of d explanatory variables for observation y_i with $x_1 \in \mathcal{X}_1, \ldots, x_d \in \mathcal{X}_d$, the independent random errors $\boldsymbol{\varepsilon} = (\varepsilon_1, \ldots, \varepsilon_n)$ have mean 0 and constant variance σ^2, and f is defined on the product domain $\mathcal{X} = \mathcal{X}_1 \times \mathcal{X}_2 \times \cdots \times \mathcal{X}_d$. The domains $\mathcal{X}_1, \ldots, \mathcal{X}_d$ are arbitrary sets. Let $\mathcal{H}^{(k)}$ be the RKHS on $\mathcal{X}_k$ with the RK $R^{(k)}$, then there exists a RHKS $\mathcal{H}$ on $\mathcal{X}$, whose RK equals

$$R(\boldsymbol{x}, \boldsymbol{z}) = R^{(1)}(x_1, z_1)R^{(2)}(x_2, z_2) \cdots R^{(d)}(x_d, z_d), \tag{1.23}$$

where $\boldsymbol{x} = (x_1, \ldots, x_d) \in \mathcal{X}$ and $\boldsymbol{z} = (z_1, \ldots, z_d) \in \mathcal{X}$. This $\mathcal{H}$ is called the tensor product RKHS, and is denoted as $\mathcal{H} = \mathcal{H}^{(1)} \otimes \mathcal{H}^{(2)} \otimes \cdots \otimes \mathcal{H}^{(d)}$. Then, given the tensor sum decomposition of each marginal space $\mathcal{H}^{(k)} = \mathcal{H}_0^{(k)} \oplus \mathcal{H}_1^{(k)}$ with $\mathcal{H}_0^{(k)} = \text{span}\{1\}$, we have

$$\mathcal{H} = \bigotimes_{k=1}^{d} \mathcal{H}^{(k)}$$

$$
\begin{aligned}
&= \bigotimes_{k=1}^{d} \left(\mathcal{H}_0^{(k)} \oplus \mathcal{H}_1^{(k)} \right) \\
&= \bigoplus_{B \subseteq \{1,\ldots,d\}} \left\{ \left(\bigotimes_{k \in B} \mathcal{H}_1^{(k)} \right) \otimes \left(\bigotimes_{k \in B^c} \mathcal{H}_0^{(k)} \right) \right\} \\
&= \bigoplus_{B \subseteq \{1,\ldots,d\}} \mathcal{H}_B \\
&= \mathcal{H}_\emptyset \oplus \sum_{k=1}^{d} \mathcal{H}_{\{k\}} \oplus \sum_{k<\ell} \mathcal{H}_{\{k,\ell\}} \oplus \cdots \oplus \mathcal{H}_{\{1,\ldots,d\}}, \qquad (1.24)
\end{aligned}
$$

where $\mathcal{H}_\emptyset = \bigotimes_{k=1}^{d} \mathcal{H}_0^{(k)}$. Therefore, for $f \in \mathcal{H}$, it can be decomposed accordingly:

$$
f = \mu + \sum_{k=1}^{d} f_k(x_k) + \sum_{k<\ell} f_{k,\ell}(x_k, x_\ell) + \cdots + f_{1,\ldots,d}(x_1, \ldots, x_d), \qquad (1.25)
$$

where $\mu \in \mathcal{H}_\emptyset$ is the overall mean, $f_k(x_k) \in \mathcal{H}_{\{k\}}$ is the main effect of x_k, $f_{k,\ell}(x_k, x_\ell) \in \mathcal{H}_{\{k,\ell\}}$ is the two-way interaction between x_k and x_ℓ, and so on. Because the similarity to the classical ANOVA model, (1.25) is called the SS-ANOVA decomposition.

In practice, spaces containing high-order interactions are usually omitted to overcome the curse of dimensionality problem. For example, if we assume $f \in \mathcal{H}_\emptyset \oplus \sum_{k=1}^{d} \mathcal{H}_{\{k\}}$, i.e., no interactions are considered, then we have the additive model (Hastie and Tibshirani (1990)). An SS-ANOVA model is referred to a model that contains any subset of the components in the SS-ANOVA decomposition. After regrouping terms, the model space of an SS-ANOVA model can be written as

$$
\mathcal{Q} = \mathcal{H}^0 \oplus \mathcal{H}^1 \oplus \cdots \oplus \mathcal{H}^q, \qquad (1.26)
$$

where $\mathcal{H}^0$ contains all functional components that are not penalized, and $\mathcal{H}^1, \ldots, \mathcal{H}^q$ are orthogonal RKHS's with RK R^j for $j = 1, \ldots, q$. Then for each $\boldsymbol{\lambda} = (\lambda_1, \ldots, \lambda_q)'$, the

estimate $\hat{f}_{\boldsymbol{\lambda}}$ in the SS-ANOVA model is

$$\hat{f}_{\boldsymbol{\lambda}} = \underset{f \in \mathcal{Q}}{\operatorname{argmin}} \left(\frac{1}{n} \sum_{i=1}^{n} \big(y_i - f(\boldsymbol{x}_i)\big)^2 + \sum_{j=1}^{q} \lambda_j \|P_j f\|^2 \right), \tag{1.27}$$

where each $\mathcal{H}^j$ has its own smoothing parameter λ_j, and P_j is the orthogonal projector in $\mathcal{Q}$ onto $\mathcal{H}^j$. Let $\mathcal{H}_1^* = \mathcal{H}^1 \oplus \cdots \oplus \mathcal{H}^q$, then for $f \in \mathcal{H}_1^*$, we have $f(\boldsymbol{x}) = \sum_{j=1}^{q} f_j(\boldsymbol{x})$ where $f_j \in \mathcal{H}^j$ for $j = 1, \ldots, q$. After the reparameterization $\lambda_j \triangleq \lambda/\theta_j$, the minimization problem in (1.27) is equivalent to

$$\hat{f}_{\lambda,\boldsymbol{\theta}} = \underset{f \in \mathcal{Q}}{\operatorname{argmin}} \left(\frac{1}{n} \sum_{i=1}^{n} \big(y_i - f(\boldsymbol{x}_i)\big)^2 + \lambda \|P_1^* f\|_*^2 \right), \tag{1.28}$$

where $\boldsymbol{\theta} = (\theta_1, \ldots, \theta_q)'$, $P_1^* = \sum_{j=1}^{q} P_j$ is the orthogonal projector in $\mathcal{Q}$ onto $\mathcal{H}_1^*$, and $\mathcal{H}_1^*$ is the RKHS with the RK $R_1^* = \sum_{j=1}^{q} \theta_j R^j$ and inner product $\langle f, g \rangle_* = \sum_{j=1}^{q} \theta_j^{-1} \langle f_j, g_j \rangle$. Because (1.28) has the same form as (1.8), the same technique can be used to solve $\hat{f}_{\lambda,\boldsymbol{\theta}}$ for fixed λ and $\boldsymbol{\theta}$. The solution $\hat{f}_{\lambda,\boldsymbol{\theta}}$ has a similar representation to (1.18), by the Kimeldorf–Wahba representer theorem:

$$\hat{f}_{\lambda,\boldsymbol{\theta}}(\boldsymbol{x}) = \sum_{\nu=1}^{p} d_\nu \phi_\nu(\boldsymbol{x}) + \sum_{i=1}^{n} c_i \sum_{j=1}^{q} \theta_j R^j(\boldsymbol{x}, \boldsymbol{x}_i), \tag{1.29}$$

where $\{\phi_1, \ldots, \phi_p\}$ is the basis of $\mathcal{H}^0$. Again, criteria like GCV can be used to select the tuning parameters λ and $\boldsymbol{\theta}$. See Gu (2013b) and Wang (2011).

1.3 COmponent Selection and Smoothing Operator

For the SS ANOVA model, we can see that the representation in (1.29) contains the representers from each $\mathcal{H}^j$ for $j = 1, \ldots, q$. Often it is necessary to perform variable selection and leave out some SS-ANOVA components because the corresponding variables are

not important. This is the motivation behind the COmponent Selection and Smoothing Operator (COSSO) method proposed in Lin and Zhang (2006).

Consider the same tensor product decomposition in (1.24), where each marginal space $\mathcal{H}^{(k)}$ is decomposed into $\{1\} \oplus \mathcal{H}_1^{(k)}$. Then for the model space $\mathcal{Q}$ in (1.26), the null space is $\mathcal{H}^0 = \text{span}\{1\}$. The COSSO procedure estimates $f \in \mathcal{Q}$ using

$$\hat{f}_\tau = \underset{f \in \mathcal{Q}}{\text{argmin}} \frac{1}{n} \sum_{i=1}^{n} \big(y_i - f(\boldsymbol{x}_i)\big)^2 + \tau^2 \sum_{j=1}^{q} \|P_j f\|, \tag{1.30}$$

where τ is the tuning parameter. The constant is the only term not penalized. Parametric components such as low order polynomials are also adaptively selected in COSSO. Comparing (1.30) with the SS-ANOVA estimate in (1.27), we can see there are two differences. First, there is only one tuning parameter in (1.30), whereas in (1.27) there are q of them, one for each $\mathcal{H}^j$. Second, the penalty term in (1.30) is the sum of RKHS norms, but in (1.27) the sum of squared RKHS norms is used instead. The reason that the penalty $\|P_j f\|$ is employed in COSSO is to induce sparsity of the SS-ANOVA components, similar to the L_1 penalty used in the LASSO model (see next section). Indeed, the LASSO is a special case of COSSO when the linear models are considered.

Instead of solving (1.30) directly, the COSSO procedure uses the following equivalent formulation:

$$\hat{f}_\lambda = \underset{f \in \mathcal{Q}}{\text{argmin}} \frac{1}{n} \sum_{i=1}^{n} \big(y_i - f(\boldsymbol{x}_i)\big)^2 + \lambda_0 \sum_{j=1}^{q} \theta_j^{-1} \|P_j f\|^2 + \lambda \sum_{j=1}^{q} \theta_j, \tag{1.31}$$

where $\theta_j \geq 0$ for $j = 1, \ldots, q$, λ_0 is a constant, and λ is now the tuning parameter. The advantage of using (1.31) is that it is very similar to the SS-ANOVA formulation in (1.27), except there is an extra penalty on the θ_j's to induce the sparsity of the components f_j's. When θ_j's are fixed, (1.31) is exactly the same minimization problem

as in the SS-ANOVA model. Thus, the COSSO algorithm first initializes $\theta_j = 1$ for $j = 1, \ldots, q$, and obtains the SS-ANOVA estimate of f with λ_0 tuned by using either CV or GCV criterion. Denoting this estimate as $\hat{f}_{SS}$, then it has similar representation as in (1.29):

$$\hat{\boldsymbol{f}}_{SS} = \hat{d}\,\mathbf{1}_n + \boldsymbol{\Sigma}_{\boldsymbol{\theta}}\hat{\boldsymbol{c}}, \tag{1.32}$$

where $\mathbf{1}_n$ is the column vector consisting of n 1's, $\hat{\boldsymbol{c}} = (\hat{c}_1, \ldots, \hat{c}_n)'$, $\boldsymbol{\Sigma}_{\boldsymbol{\theta}} = \sum_{k=1}^{q} \theta_k \boldsymbol{\Sigma}_k$ with $\boldsymbol{\Sigma}_k = \left\{R^k(x_i, x_j)\right\}_{i,j=1}^{n}$, $\hat{\boldsymbol{f}}_{SS} = \left(\hat{f}_{SS}(x_1), \ldots, \hat{f}_{SS}(x_n)\right)'$. At the next step, the COSSO algorithm solves the following non-negative garrote problem to obtain the estimates of $\boldsymbol{\theta} = (\theta_1, \ldots, \theta_q)'$ with fixed $\hat{d}$ and $\hat{\boldsymbol{c}}$.

$$\min_{\boldsymbol{\theta}} (\boldsymbol{z} - \boldsymbol{G}\boldsymbol{\theta})'(\boldsymbol{z} - \boldsymbol{G}\boldsymbol{\theta}) \quad \text{subject to } \theta_j \geq 0, j = 1, \ldots, q; \sum_{j=1}^{q} \theta_j \leq M, \tag{1.33}$$

where $\boldsymbol{z} = \boldsymbol{y} - \frac{1}{2} n \lambda_0 \hat{\boldsymbol{c}} - \hat{d}\mathbf{1}_n$, $\boldsymbol{G} = (\boldsymbol{\Sigma}_1\hat{\boldsymbol{c}}, \ldots, \boldsymbol{\Sigma}_q\hat{\boldsymbol{c}})$ is an $n \times q$ matrix, and the tuning parameter M is selected by CV or GCV criterion. Denote the the estimate of $\boldsymbol{\theta}$ as $\hat{\boldsymbol{\theta}}$. Then at the final step, the COSSO algorithm solves (1.31) again with fixed $\boldsymbol{\theta} = \hat{\boldsymbol{\theta}}$, to obtain the final estimate $\hat{\boldsymbol{f}}_{\hat{\boldsymbol{\theta}}}$. Lin and Zhang (2006) implemented the COSSO using $\mathcal{H}^{(k)} = W_2^2[0, 1]$ for $k = 1, \ldots, d$.

1.4 LASSO and Adaptive LASSO

1.4.1 Least Absolute Shrinkage and Selection Operator (LASSO)

Consider the same regression problem as in (1.22) with $f(\boldsymbol{x}_i) = \sum_{j=1}^{d} x_{ij}\beta_j$ and $\mathcal{X} = \mathbb{R}^d$. Also assume x_{ij} are standardized so that $\sum_{i=1}^{n} x_{ij}/n = 0$ and $\sum_{i=1}^{n} x_{ij}^2 = 1$. Let $\boldsymbol{\beta} = (\beta_1, \ldots, \beta_d)'$ be the coefficients vector. Then the LASSO estimate of $\boldsymbol{\beta}$ is the

solution to the following L_1-penalized regression problem:

$$\hat{\boldsymbol{\beta}}_{\lambda}^{\text{lasso}} = \underset{\boldsymbol{\beta}}{\operatorname{argmin}} \left(\frac{1}{2} \sum_{i=1}^{n} \left(y_i - \sum_{j=1}^{d} x_{ij} \beta_j \right)^2 + \lambda \sum_{j=1}^{d} |\beta_j| \right), \tag{1.34}$$

where λ is the tuning parameter that controls the shrinkage of all the coefficients. As λ increases, LASSO continuously shrinks the coefficients towards zero, and some coefficients are shrunk to exact zero when λ is sufficiently large. The solution in (1.34) has closed form only for some special cases. For example, when there is a single predictor, i.e., $d = 1$, the LASSO solution is a soft-thresholded version of the least square estimate $\hat{\beta}^{\text{ols}}$:

$$\begin{aligned} \hat{\beta}_{\lambda}^{\text{lasso}} &= S(\hat{\beta}^{\text{ols}}, \lambda) \triangleq \text{sign}(\hat{\beta}^{\text{ols}})(|\hat{\beta}^{\text{ols}}| - \lambda)_+ \\ &= \begin{cases} \hat{\beta}^{\text{ols}} - \lambda, & \text{if } \hat{\beta}^{\text{ols}} > 0 \text{ and } \lambda < |\hat{\beta}^{\text{ols}}|, \\ \hat{\beta}^{\text{ols}} + \lambda, & \text{if } \hat{\beta}^{\text{ols}} < 0 \text{ and } \lambda < |\hat{\beta}^{\text{ols}}|, \\ 0, & \text{if } \lambda \geq |\hat{\beta}^{\text{ols}}|, \end{cases} \end{aligned} \tag{1.35}$$

where the least square estimate is $\hat{\beta}^{\text{ols}} = \sum_{i=1}^{n} x_i y_i$. With multiple predictors, if they are uncorrelated, or equivalently when the design matrix is orthonormal, i.e., $\boldsymbol{X}'\boldsymbol{X} = \boldsymbol{I}$, then once again the LASSO estimate of each β_j is the soft-thresholded version of the least square estimate $\hat{\beta}_j^{\text{ols}}$. For the special case in (1.35), we can see the continuous shrinkage nature of the LASSO estimator, which helps to improve the prediction accuracy due to the bias-variance trade-off. This trade-off is controlled by λ. By allowing small increase in the bias in exchange of substantial decrease in the variance when estimating $\boldsymbol{\beta}$, the prediction error can be reduced substantially.

The pathwise coordinate descent algorithm proposed in Friedman et al. (2007) can be used to efficiently solve (1.34). The idea is to repeatedly apply soft-thresholding with a "partial residual" as a response variable. We can rewrite the right hand side of (1.34)

as:

$$g(\tilde{\boldsymbol{\beta}}) = \frac{1}{2}\sum_{i=1}^{n}\left(y_i - \sum_{k\neq j} x_{ik}\tilde{\beta}_k - x_{ij}\beta_j\right)^2 + \lambda\sum_{k\neq j}|\tilde{\beta}_k| + \lambda|\beta_j|, \tag{1.36}$$

where all the values of β_k for $k \neq j$ are fixed at prior iterations' values $\tilde{\beta}_k(\lambda)$, and we update the estimate of a single coefficient β_j by minimizing (1.36). Based on the solution in (1.35), it is clear that we get

$$\tilde{\beta}_j(\lambda) \leftarrow S\left(\sum_{i=1}^{n} x_{ij}\big(y_i - \tilde{y}_i^{(j)}\big),\ \lambda\right), \tag{1.37}$$

where $\tilde{y}_i^{(j)} = \sum_{k\neq j} x_{ik}\tilde{\beta}_k$. The update (1.37) is repeated for $j = 1, 2, \ldots, d, 1, 2, \ldots$ until convergence. The pathwise coordinate descent algorithm returns the LASSO solution for any fixed λ. To tune λ, a criterion such as GCV can be used.

1.4.2 Adaptive LASSO

To evaluate the performance of a variable selection procedure, Fan and Li (2001) proposed the oracle properties. The properties have two parts: one is the consistency of variable selection, i.e., a procedure can correctly identify the true nonzero coefficients; the other is the optimal root-n estimation rate of these true nonzero coefficients. Zou (2006) has shown that the LASSO does not always satisfy the oracle properties, and thus proposed a new procedure called the adaptive LASSO to achieve oracle properties.

The idea of the adaptive LASSO is to penalize the coefficients differently by assigning different weights to them. The adaptive LASSO estimator, $\hat{\boldsymbol{\beta}}^{\mathrm{AL}}$, can be obtained by solving the following L_1-penalized regression problem:

$$\hat{\boldsymbol{\beta}}_{\lambda}^{\mathrm{AL}} = \underset{\boldsymbol{\beta}}{\operatorname{argmin}}\left(\frac{1}{2}\sum_{i=1}^{n}\left(y_i - \sum_{j=1}^{d} x_{ij}\beta_j\right)^2 + \lambda\sum_{j=1}^{d}\hat{w}_j|\beta_j|\right), \tag{1.38}$$

where $\hat{\boldsymbol{w}} = (\hat{w}_1, \ldots, \hat{w}_d)'$ is a data-dependent weights vector defined as $\hat{\boldsymbol{w}} = 1/|\hat{\boldsymbol{\beta}}|^\gamma$, for a root-n-consistent estimator $\hat{\boldsymbol{\beta}}$ to $\boldsymbol{\beta}$, and a positive γ. Usually we use $\hat{\boldsymbol{w}} = 1/|\hat{\boldsymbol{\beta}}^{\text{ols}}|^\gamma$.

The algorithm for obtaining adaptive LASSO estimates in (1.38) is simply based on the LASSO algorithm:

The Adaptive LASSO Algorithm

1. Let $x^*_{ij} = x_{ij}/\hat{w}_j$, for $i = 1, \ldots, n$, $j = 1, \ldots, d$, where $\hat{w}_j = 1/|\hat{\beta}^{\text{ols}}_j|^\gamma$ for fixed γ.

2. Solve the LASSO problem for fixed λ:

$$\hat{\boldsymbol{\beta}}^* = \underset{\boldsymbol{\beta}}{\text{argmin}} \frac{1}{2} \sum_{i=1}^{n} \left(y_i - \sum_{j=1}^{d} x^*_{ij} \beta_j \right)^2 + \lambda \sum_{j=1}^{d} |\beta_j|. \tag{1.39}$$

3. The adaptive LASSO estimates for fixed λ and γ are:

$$\hat{\beta}^{\text{AL}}_j = \hat{\beta}^*_j/\hat{w}_j, \quad j = 1, \ldots, d. \tag{1.40}$$

The adaptive LASSO has two tuning parameters: λ and γ. To find an optimal pair of (λ, γ), a method such as cross-validation can be used.

1.5 Hybrid Adaptive Spline

The smoothing spline, smoothing spline ANOVA, and COSSO are regularization approaches. They assume f is in certain model space, and then each solve a regularized least square problem by penalizing the model complexity. In contrast, the regression spline is a basis approximation approach. It starts with representing f in terms of basis functions, such as in (1.2), and then selects the optimal number and types of basis accord-

ing to certain model selection criterion. The hybrid adaptive spline (HAS) is proposed in Luo and Wahba (1997), and it can also be classified as a basis approximation (pursuit) approach. It is called "hybrid" because it is similar to the regression spline, but with smoothing spline representers as basis functions.

When $f \in W_2^m[0,1]$, the smoothing spline estimator of f has the representation in (1.18), which is a linear combination of the basis in $\mathcal{H}_0$ and the representers in $\mathcal{H}_1$. The HAS idea is to select only k representers adaptively among $\mathcal{R}_1 = \{\xi_1(x), \ldots, \xi_n(x)\}$ to approximate f. This is useful especially when the underlying function is spatially inhomogeneous. The tuning parameter k controls the model complexity, and can be selected using the GCV criterion. Luo and Wahba (1997) presented the HAS procedure only for the $m = 2$ case, but the procedure is similar for general m, which is presented as follows.

The HAS procedure

1. *Initialization:* Set $\mathcal{B}_m = \{\phi_1(x), \ldots, \phi_m(x)\}$ and let M be an upper bound on the number of basis functions to be selected (including those in $\mathcal{B}_m$).

2. *Forward Selection:* For $k = m+1, \ldots, M$, select ξ_{j_k} among the representers that have not yet been selected, to maximize the reduction of RSS:

$$\xi_{j_k} = \underset{\xi \in \mathcal{R}_1 \cap \mathcal{B}_{k-1}^c}{\operatorname{argmax}} \Big\{ \mathrm{RSS}\big(\mathcal{B}_{k-1}\big) - \mathrm{RSS}\big(\mathcal{B}_{k-1} \cup \{\xi\}\big) \Big\}, \tag{1.41}$$

   where $\mathcal{B}_{k-1}$ is the set that contains all the basis functions in $\mathcal{B}_m$ and the $(k-1-m)$ representers that have been selected in prior steps, $\mathrm{RSS}(\mathcal{B})$ is the RSS of the ordinary least squares fit of the model with all the basis functions in $\mathcal{B}$ as covariates. Update $\mathcal{B}_k = \mathcal{B}_{k-1} \cup \{\xi_{j_k}\}$.

3. *Elimination:* Select k^*, $m \leq k^* \leq M$, to minimize the GCV criterion:

$$\text{GCV}(k) = \frac{\text{RSS}(\mathcal{B}_k)}{\left[n - (m + (k-m) \times \text{IDF})\right]^2}, \tag{1.42}$$

   where IDF is the inflated degrees of freedom.

4. *Final Model:* Fit an model with ordinary least squares or ridge regression method using the basis functions contained in $\mathcal{B}_{k^*}$.

In the calculation of GCV in (1.42), the degrees of freedom of each adaptively selected representers are inflated by a factor of IDF (hence the name inflated degrees of freedom) to account for the adaptive model selection process. Based on simulations, the authors suggested that 1.2 is a good choice for the IDF in HAS, when $m = 2$. A larger IDF may cause fewer basis functions to be selected in the final model. One potential drawback of the HAS procedure is due to this ad hoc choice of IDF. We will delay the discussions of the IDF until Section 1.7. Another drawback of HAS comes from the fact that it uses only one type of basis functions — the smoothing spline representers — in the forward selection. These two drawbacks are the main motivations for the BSML procedure, which we will discuss in Section 1.8. In addition, the search procedure in HAS is greedy which may be trapped to a local optima. We will propose a look-ahead search procedure in Chapter 3 to alleviate this drawback.

1.6 Multivariate Adaptive Regression Splines

Multivariate Adaptive Regression Splines (MARS) is another basis approximation approach that was proposed in Friedman (1991). It assumes that f is a linear combination of tensor products of univariate linear spline basis functions.

Consider the same regression problem as in (1.22) with $\mathcal{X} = \mathbb{R}^d$, and a point $\boldsymbol{X} \in \mathcal{X}$

is written as $\boldsymbol{X} = (X_1, \ldots, X_d)$. Let $\mathcal{T}_1$ be the collection of pairs of truncated linear functions:

$$\mathcal{T}_1 = \left\{(X_j - t)_+,\, (t - X_j)_+\right\}_{\substack{t\in\{x_{1j},x_{2j},\ldots,x_{nj}\}\\ j=1,2,\ldots,d.}}, \tag{1.43}$$

where the knot t is one of the observed values of the jth explanatory variable X_j, and X_j ranges over $(-\infty, \infty)$. If all the observed values of the jth explanatory variable are distinct for each $j \in \{1, \ldots, d\}$, then there are $2dn$ basis functions in $\mathcal{T}_1$. Note that each basis $h(\boldsymbol{X}) = (X_j - t)_+$ is considered as a function in the entire input space $\mathbb{R}^d$, although it only depends on a single variable X_j. MARS assumes f has the following representation:

$$f(\boldsymbol{X}) = \beta_0 + \sum_{m=1}^{M} \beta_m h_m(\boldsymbol{X}), \tag{1.44}$$

where $h_m(\boldsymbol{X})$ is either a function in $\mathcal{T}_1$ or a product of two or more such functions, and M controls the complexity of f and is user-specified. The advantage of using linear splines in $\mathcal{T}_1$ as building blocks is that now h_m is nonzero only over the small part of the input space, where all its components are nonzero. This allows the h_m basis functions to operate locally and build up a parsimonious regression surface.

Given the basis functions, the coefficients β_m's are estimated using ordinary least squares. To construct the set h_m's, MARS first performs forward selection. Starting with the constant $h_0(\boldsymbol{X}) = 1$, suppose at the kth step the model set is $\mathcal{M}_k$ which contains all the bases selected in the prior steps, then MARS computes the reduction of RSS by adding to $\mathcal{M}_k$ the products of each basis in $\mathcal{M}_k$ with every pair $\left\{(X_j - t)_+,\, (t - X_j)_+\right\}$ in $\mathcal{T}_1$ that has not yet been selected, and then selects the pair that minimizes the reduction of RSS. The forward selection stops when adding new terms will make the total number of terms in the model exceed $M + 1$. A backward deletion procedure is then applied, deleting the term one by one whose removal causes the smallest increase in RSS, to produce a sequence of candidate models $\hat{f}_\lambda$ of each size λ. Denote d as the total number

of terms in the model when the forward selection is completed, where $d \leq M+1$, then the size $\lambda \in \{d, d-1, \ldots, 2\}$. The final model is $\hat{f}_{\lambda^*}$ where λ^* is chosen to minimize the GCV criterion. Here are the details of the MARS algorithm.

The MARS procedure

1. *Initialization:* Set $\mathcal{B}_1 = \mathcal{M}_1 = \{1\}$ and $k = 1$. We use $\mathcal{B}$ to denote the set of basis functions selected in the forward selection, and $\mathcal{M}$ to denote the set of model terms h_m's induced by these basis functions. Let M be the user-specified maximum number of model terms (does not include the constant term) as in (1.44).

2. *Forward Selection:*

   while $|\mathcal{M}_k| < M+1$ **do**

$$\begin{aligned}
&\big\{(X_{j^*} - t^*)_+,\, (t^* - X_{j^*})_+\big\} \\
&= \underset{\substack{(X_j - t)_+ \in (\mathcal{T}_1 \cap \mathcal{B}_k^c) \\ (t - X_j)_+ \in (\mathcal{T}_1 \cap \mathcal{B}_k^c)}}{\operatorname{argmax}} \Big\{ \mathrm{RSS}(\mathcal{M}_k) - \mathrm{RSS}\Big(\mathcal{M}_k \cup \bigcup_{h_\ell \in \mathcal{M}_k} \big\{(X_j - t)_+ \cdot h_\ell(\boldsymbol{X}), \\
&\qquad\qquad (t - X_j)_+ \cdot h_\ell(\boldsymbol{X})\big\}\Big)\Big\}
\end{aligned}$$

   Update:

$$\begin{aligned}
\mathcal{B}_{k+1} &= \mathcal{B}_k \cup \big\{(X_{j^*} - t^*)_+,\, (t^* - X_{j^*})_+\big\}, \\
\mathcal{M}_{k+1} &= \mathcal{M}_k \cup \bigcup_{h_\ell \in \mathcal{M}_k} \big\{(X_{j^*} - t^*)_+ \cdot h_\ell(\boldsymbol{X}), (t^* - X_{j^*})_+ \cdot h_\ell(\boldsymbol{X})\big\}, \\
k &= k + 1.
\end{aligned}$$

   end while

3. *Backward Elimination:* Let $\mathcal{M}_f$ be the model set returned from the forward

selection. Let $d = |\mathcal{M}_f|$. Set $\mathcal{M}^{(d)} = \mathcal{M}_f$.

for $\lambda = d, d-1, \ldots, 2$ **do**

$$\xi^{\lambda} = \underset{\xi \in \mathcal{M}^{(\lambda)}}{\operatorname{argmin}} \Big\{ \mathrm{RSS}\big(\mathcal{M}^{(\lambda)} \setminus \{\xi\}\big) - \mathrm{RSS}(\mathcal{M}^{(\lambda)}) \Big\}.$$

$$\text{Set} \quad \mathcal{M}^{(\lambda-1)} = \mathcal{M}^{(\lambda)} \setminus \{\xi^{\lambda}\}.$$

end for

Select λ^*, $1 \leq \lambda^* \leq d$, to minimize the GCV criterion:

$$\mathrm{GCV}(\lambda) = \frac{\sum_{i=1}^{n} (y_i - \hat{f}_{\lambda}(\boldsymbol{x_i}))^2}{\big[n - (r_{\lambda} + K_{\lambda} \times \mathrm{IDF})\big]^2}, \tag{1.45}$$

where $\hat{f}_{\lambda}$ is the ordinary least square fit using the basis functions in $\mathcal{M}^{(\lambda)}$, r_{λ} is the number of terms in $\mathcal{M}^{(\lambda)}$, K_{λ} is the number of knots in $\mathcal{M}^{(\lambda)}$, and IDF is the inflated degrees of freedom. For MARS, Friedman suggested to use IDF = 3.

4. *Final Model:* $\hat{f}_{\lambda^*}$.

The advantage of MARS is that it can be easily applied to high-dimensional data. Also, it allows high-order interactions among the predictor variables, and is very useful when the true model has a hierarchical structure. However, MARS shares the same disadvantages as HAS. They both use fixed IDF, single class of basis (linear splines in MARS), and have a greedy search procedure.

1.7 Generalized Degrees of Freedom and Covariance Penalty

1.7.1 Ideas Behind the IDF and GDF

We have seen in Sections 1.5 and 1.6 that the IDF plays an important role in constructing the model selection criterion in each of HAS and MARS. The concepts of IDF and generalized degrees of freedom (GDF) are also crucial in understanding other adaptive basis selection procedures such as the BSML procedure (Sklar et al. (2013)) and the look-ahead procedure discussed later. For example, during the forward selection stage of the look-ahead procedure, basis functions are selected by taking into account the IDFs of different libraries. The basis functions are grouped into libraries according to certain properties, such as the form of the basis function or smoothness of this function. Previous literature has suggested that it is advantageous to use different IDFs for basis functions that possess different properties. As mentioned earlier, in the HAS procedure (where the bases are smoothing spline representers), Luo and Wahba (1997) suggested IDF=1.2; while for the MARS procedure (which has the truncated linear functions as its basis functions), Friedman (1991) suggested IDF=3. The potential problem with fixing the IDF is that even if only one library of basis functions is used, the IDF should change as more and more bases are selected, because the IDF should depend on both the number of bases already in the current model, and how well this model fits the data. Therefore, it is restrictive to fix the IDF for each library throughout the searching process. In addition, the choices of IDF in HAS and MARS are ad hoc. The IDF should be chosen adaptively based on the data. For a particular data set, the degrees of freedom of certain basis functions should be inflated less (making them easier to select) compared to the others, when they possess key properties that lead to better approximation of the true function.

However, the situation may change completely for another data set with a different true function. Therefore, the estimation of the IDFs should be data-driven and adaptive. This is also the idea behind the BSML procedure which estimates IDF values based on an estimate of generalized degrees of freedom.

Generalized degrees of freedom (GDF) was proposed in Ye (1998) for Gaussian data, where it was defined as the the sum of the sensitivities of the fitted values to perturbations in the response. Here we summarize Ye's approach. Let $\boldsymbol{y} = (y_1, \ldots, y_n)'$ be the observed values of a response variable Y, $\boldsymbol{f} = \big(f(\boldsymbol{x}_1), \ldots, f(\boldsymbol{x}_n)\big)'$ be the values of the true function at the design points $\boldsymbol{x}_1, \ldots, \boldsymbol{x}_n$. The response vector $\boldsymbol{y}$ is assumed to follow a Gaussian distribution:

$$\boldsymbol{y} \sim N(\boldsymbol{f},\ \sigma^2 \boldsymbol{I}). \tag{1.46}$$

Also, let $\hat{f}$ be the estimate of the true function f, and $\hat{\boldsymbol{f}} = \big(\hat{f}(\boldsymbol{x}_1), \ldots, \hat{f}(\boldsymbol{x}_n)\big)'$. A modeling procedure $\mathcal{M}$ is defined as the mapping from $\boldsymbol{y}$ to $\hat{\boldsymbol{f}}$:

$$\mathcal{M} : \boldsymbol{y} \to \hat{\boldsymbol{f}}. \tag{1.47}$$

Then the GDF of $\mathcal{M}$ is defined as

$$\begin{aligned} D(\mathcal{M}) &\triangleq \sum_{i=1}^{n} \frac{\partial \mathbb{E}_{\boldsymbol{f}}(\hat{f}(\boldsymbol{x}_i))}{\partial f(\boldsymbol{x}_i)} \\ &= \frac{1}{\sigma^2} \sum_{i=1}^{n} \mathrm{Cov}\left(\hat{f}(\boldsymbol{x}_i),\ y_i - f(\boldsymbol{x}_i)\right) \\ &= \frac{1}{\sigma^2} \sum_{i=1}^{n} \mathrm{Cov}\left(\hat{f}(\boldsymbol{x}_i),\ y_i\right). \end{aligned} \tag{1.48}$$

GDF provides a good measure of the model complexity and reduces to the regular degrees of freedom in Gaussian linear regression with given covariates, but its estimation can be computationally expensive in more complicated models. Ye (1998) proposed the

Monte Carlo algorithm to estimate the GDF when it does not have a known analytical value. First, T replicates of Gaussian noises $\boldsymbol{\delta}_t$ (called the perturbations) are simulated, i.e.,

$$\boldsymbol{\delta}_t = (\delta_{t1}, \ldots, \delta_{tn}) \sim N(\mathbf{0},\ \tau^2 \boldsymbol{I}_n),\ \text{for } t = 1, \ldots, T, \tag{1.49}$$

where $\tau \in [0.5\hat{\sigma}, \hat{\sigma}]$ and $\hat{\sigma}$ is an estimate of σ. Then the modeling procedure $\mathcal{M}$ is applied using each perturbed response $\boldsymbol{y} + \boldsymbol{\delta}_t$ to obtain the estimate $\hat{f}_{\boldsymbol{y}+\boldsymbol{\delta}_t}$, for $t = 1, \ldots, T$. Finally, for fixed $\boldsymbol{x}_i$, for each of $i = 1, \ldots, n$, the sensitivity $\hat{h}_i$ is calculated as the least-squares estimate of slope in the following model

$$\hat{f}_{\boldsymbol{y}+\boldsymbol{\delta}_t}(\boldsymbol{x}_i) = \hat{\alpha} + \hat{h}_i \delta_{ti}, \quad t = 1, \ldots, T, \tag{1.50}$$

and the estimate of GDF is $\hat{D}(\mathcal{M}) = \sum_{i=1}^{n} \hat{h}_i$. Therefore one needs to fit the model $\mathcal{M}$ T times to compute this estimate of the GDF.

The BSML procedure estimates the GDF of the models at each step while performing the forward selection. Let $\hat{D}(\mathcal{M}_k)$ be the estimated GDF of the model $\mathcal{M}_k$ which contains k basis functions, then the IDF of this model is estimated by

$$\widehat{\mathrm{IDF}}(\mathcal{M}_k) = \frac{\hat{D}(\mathcal{M}_k) - m}{k - m}, \tag{1.51}$$

where the IDFs of the first m bases are fixed at 1.

In the next section, we will explain the relationship between GDF, RSS, and MSE.

1.7.2 The Relationship Between GDF, RSS, and MSE

Consider model (1.22). Let $\mathcal{M}$ be the modeling procedure that leads to an estimate $\hat{\boldsymbol{f}}$. Define the covariance penalty $C(\mathcal{M})$ as

$$C(\mathcal{M}) \triangleq \mathbb{E}\big[(\boldsymbol{y}-\boldsymbol{f})'(\hat{\boldsymbol{f}}-\boldsymbol{f})\big] = \sum_{i=1}^{n} \mathrm{Cov}\big(\hat{f}(\boldsymbol{x}_i), y_i\big).$$

Let's take a closer look at the covariance penalty $C(\mathcal{M})$. In the standard linear regression setting, conditional on including all of a fixed set of p covariates (analogous to p basis functions evaluated at n discrete points), the $n \times p$ design matrix $\boldsymbol{X}$ is fixed. Hence the modeling procedure $\mathcal{M}$ will only include the model estimation part, since no selection of the covariates is needed. Let $\boldsymbol{H}$ be the corresponding hat matrix, i.e., $\hat{\boldsymbol{y}} = \hat{\boldsymbol{f}} = \boldsymbol{H}\boldsymbol{y}$. Then $C(\mathcal{M})$ has the following explicit form:

$$\begin{aligned}
C(\mathcal{M}) &\triangleq \mathbb{E}\big[(\boldsymbol{y}-\boldsymbol{f})'(\hat{\boldsymbol{f}}-\boldsymbol{f})\big] \\
&= \mathbb{E}\big\{(\boldsymbol{y}-\boldsymbol{f})'\hat{\boldsymbol{f}}\big\} - \big(\underbrace{\mathbb{E}(\boldsymbol{y})-\boldsymbol{f}}_{=0}\big)'\boldsymbol{f} \\
&= \mathbb{E}\big\{(\boldsymbol{y}-\boldsymbol{f})'\hat{\boldsymbol{f}}\big\} \\
&= \mathbb{E}\big(\boldsymbol{\varepsilon}'\boldsymbol{H}\boldsymbol{y}\big) \\
&= \mathbb{E}\big(\boldsymbol{\varepsilon}'\boldsymbol{H}\boldsymbol{\varepsilon}\big) + \mathbb{E}\big(\boldsymbol{\varepsilon}'\boldsymbol{H}\boldsymbol{f}\big) \\
&= \mathrm{tr}(\boldsymbol{H})\sigma^2 \\
&= p\sigma^2, \qquad (1.52)
\end{aligned}$$

where the second-to-last equality is a result of Theorem 1.5 in Seber and Lee (2003). Hence, the GDF $D(\mathcal{M})$ in this linear regression case can be calculated analytically as:

$$D(\mathcal{M}) = \frac{C(\mathcal{M})}{\sigma^2} = p.$$

Because $D(\mathcal{M})$ is a function of the number of parameters p in the model, it measures the model complexity. The IDF for each basis is $D(\mathcal{M})/p = 1$ here. If, however, the modeling procedure $\mathcal{M}$ also includes the variable (basis) selection part, then the "cost" of selecting a basis function will typically be greater than 1 [1]. The actual GDF and IDF depend on many factors such as the specific modeling procedure $\mathcal{M}$, the true function, the nature of variables (basis functions), and which variables (basis functions) have been selected in prior steps. It does not have an explicit form in general.

As shown in Sklar et al. (2013), the expected value of the RSS of an model $\mathcal{B}$ can be decomposed into:

$$\begin{aligned}
\mathbb{E}(\text{RSS}) &= \mathbb{E}\left(\left\|\hat{\boldsymbol{f}} - \boldsymbol{y}\right\|^2\right) \\
&= \mathbb{E}\left(\left\|\hat{\boldsymbol{f}} - \boldsymbol{f} + \boldsymbol{f} - \boldsymbol{y}\right\|^2\right) \\
&= \mathbb{E}\left(\left\|\hat{\boldsymbol{f}} - \boldsymbol{f}\right\|^2\right) + \mathbb{E}\left(\left\|\boldsymbol{f} - \boldsymbol{y}\right\|^2\right) - 2\mathbb{E}\left[(\boldsymbol{y} - \boldsymbol{f})'(\hat{\boldsymbol{f}} - \boldsymbol{f})\right] \\
&= \text{MSE} + \mathbb{E}(\boldsymbol{\varepsilon}'\boldsymbol{\varepsilon}) - 2C(\mathcal{M}) \\
&= \text{MSE} + n\sigma^2 - 2C(\mathcal{M}).
\end{aligned}$$

That is,

$$\text{MSE} = \mathbb{E}(\text{RSS}) + 2C(\mathcal{M}) - n\sigma^2. \tag{1.53}$$

On the other hand, there is a well-known bias-variance decomposition for the MSE:

$$\begin{aligned}
\text{MSE} &= \mathbb{E}\left(\left\|\boldsymbol{f} - \hat{\boldsymbol{f}}\right\|^2\right) \\
&= \mathbb{E}\left(\left\|\boldsymbol{f} - \mathbb{E}(\hat{\boldsymbol{f}}) + \mathbb{E}(\hat{\boldsymbol{f}}) - \hat{\boldsymbol{f}}\right\|^2\right) \\
&= \left\|\boldsymbol{f} - \mathbb{E}(\hat{\boldsymbol{f}})\right\|^2 + \mathbb{E}\left(\left\|\hat{\boldsymbol{f}} - \mathbb{E}(\hat{\boldsymbol{f}})\right\|^2\right)
\end{aligned}$$

[1] For the basis functions that are always included in the model, e.g., the first m basis functions in the HAS procedure, the IDF will be 1.

$$= \left\|\text{Bias}(\hat{\boldsymbol{f}})\right\|^2 + \text{tr}\big(\text{Var}(\hat{\boldsymbol{f}})\big), \tag{1.54}$$

where $\text{Bias}(\hat{\boldsymbol{f}}) = \boldsymbol{f} - \mathbb{E}(\hat{\boldsymbol{f}})$, and $\text{Var}(\hat{\boldsymbol{f}})$ is the covariance matrix of $\hat{\boldsymbol{f}}$. For the second term in (1.54), if $\hat{\boldsymbol{f}}$ is estimated by a modeling procedure $\mathcal{M}$ that does not include selection, then

$$\begin{aligned}
\text{tr}\big(\text{Var}(\hat{\boldsymbol{f}})\big) &= \mathbb{E}\left(\left\|\hat{\boldsymbol{f}} - \mathbb{E}(\hat{\boldsymbol{f}})\right\|^2\right) \\
&= \mathbb{E}\left(\left\|\boldsymbol{H}\boldsymbol{y} - \mathbb{E}(\boldsymbol{H}\boldsymbol{y})\right\|^2\right) \\
&= \mathbb{E}\left(\left\|\boldsymbol{H}\boldsymbol{y} - \boldsymbol{H}\mathbb{E}(\boldsymbol{y})\right\|^2\right) \\
&= \mathbb{E}\left(\left\|\boldsymbol{H}\boldsymbol{y} - \boldsymbol{H}\boldsymbol{f}\right\|^2\right) \\
&= \mathbb{E}\left(\left\|\boldsymbol{H}\boldsymbol{\varepsilon}\right\|^2\right) \\
&= \mathbb{E}\left(\boldsymbol{\varepsilon}'\boldsymbol{H}'\boldsymbol{H}\boldsymbol{\varepsilon}\right) \\
&= \mathbb{E}\left(\boldsymbol{\varepsilon}'\boldsymbol{H}\boldsymbol{\varepsilon}\right), \quad \text{since } \boldsymbol{H} \text{ is idempotent,} \\
&= p\sigma^2.
\end{aligned} \tag{1.55}$$

We can see that the $C(\mathcal{M})$ in (1.52) is the same as $\text{tr}\big(\text{Var}(\hat{\boldsymbol{f}})\big)$ in (1.55) under the fixed covariates linear regression case. Therefore, (1.53) provides another view of the bias-variance trade-off. If additional basis functions are included in a model, the RSS will be smaller, but on the other hand the model's GDF will increase. An ideal linear regression model should achieve the balance between the two.

1.8 The BSML Procedure

The BSML procedure was proposed in Sklar et al. (2013). It selects basis functions adaptively from multiple libraries, where each library consists of basis functions with

similar forms and properties. Compared to using a single library, the advantage of using multiple libraries is that only relatively few basis functions need to be selected from each library to approximate the target function, particularly if the target function is spatially inhomogeneous and if the basis functions in different libraries capture different inhomogeneous features found in the true function. There are infinitely many choices of the libraries. Libraries may be selected from different families including Fourier, spline, radial, wavelet bases and so on. They may also be selected from different types within a family. We can have B-splines, truncated polynomials and reproducing kernel representers for the spline family. Within each type, we can specify different orders of basis, e.g., linear or cubic for polynomial splines.

Table 1.1 lists some commonly used libraries of basis functions (Sklar et al. (2013)). Here $\mathcal{T}_m$ are truncated polynomials, and the step functions $\mathcal{T}_0$ is a special case with $m = 0$. $\mathcal{R}_{(1,m)}$ are the polynomial spline representers defined as $\mathcal{R}_{(1,m)} = \big\{\{R_m(x_i, z_1)\}_{i=1}^n, \ldots, \{R_m(x_i, z_q)\}_{i=1}^n\big\}$ where $z_1, \ldots, z_q$ are knots. $R_m(x_i, z_j)$ is the reproducing kernel defined in (1.15). For the family $\mathcal{P}_2$, the periodic spline reproducing kernel of order-2 is defined as $R_{per,2}(s,t) = \sum_{v=1}^{\infty} \frac{2}{(2\pi v)^4} \cos 2\pi v(s-t)$ for $s, t \in [0,1]$.

Table 1.1: Notations for libraries of basis functions that will be used in Section 2.2.

Notation	Family of Basis Functions
$\mathcal{U}_0$	$\{1\}$, constant functions
$\mathcal{U}_m$	$\{1, x, x^2, \ldots, x^m\}$, polynomial functions of orders $0, 1, 2, \ldots, m$
$\mathcal{C}_m$	$\mathcal{R}_{(1,m)}$, polynomial spline representers
$\mathcal{P}_2$	$\{R_{per,2}(z_j, x)\}$, periodic smoothing spline representers of order-2
$\mathcal{T}_m$	$\{(x-z_1)_+^m, (x-z_2)_+^m, \ldots, (x-z_d)_+^m\}$, truncated ploynomials
$\mathcal{F}_{f,m}$	$\{\sin(2\pi fkx), \cos(2\pi fkx) : k = 1, \ldots, m\}$, Fourier basis functions

The BSML procedure starts with the null library $\mathcal{L}_0$, which contains all the basis functions that will be included in the model automatically. Let $m = |\mathcal{L}_0|$ and M be the

pre-specified maximum number of basis functions we want to select (including those in $\mathcal{L}_0$). Basis functions are selected from L additional libraries $\mathcal{O} = \cup_{l=1}^{L} \mathcal{L}_l$ one at a time. At each step k, denote the sequentially selected basis functions as ϕ_k for $k = m+1, \ldots, M$. Let $\mathcal{B}_k = \{\phi_1, \ldots, \phi_k\}$ for $k = m, \ldots, M$, where $\mathcal{B}_m = \mathcal{L}_0$. Write "model $\mathcal{B}_k$" for "a linear combination of the basis functions in $\mathcal{B}_k$", and also $\mathcal{M}_k$ for the modeling procedure that includes both basis functions selection and estimation steps.

There are two variations of the BSML procedures in Sklar et al. (2013): BSML-C and BSML-S. The basic idea of the BSML-S procedure is as follows: at each step k, it first selects the best basis function from each library $\mathcal{L}_l$ for $l = 1, \ldots, L$ to form a candidate set; it then selects the best basis ϕ_k among these candidates according to a criterion such as the DPC in (1.58); finally, it uses either the CIC or GCV criterion defined in (1.60) and (1.61) respectively, to select the best model $\mathcal{B}_{k^*}$, for $m \leq k^* \leq M$.

The BSML-C procdure is a simplified version of BSML-S. At each step k, it selects the best basis function from the combined non-null libraries $\mathcal{O}$. The final model $\mathcal{B}_{k^*}$ is chosen to minimize either the CIC or GCV criterion defined in (1.60) and (1.61) respectively, for $m \leq k^* \leq M$. The difference between BSML-C and BSML-S is that BSML-C does not distinguish between the selection costs of basis functions from different non-null libraries in the forward selection, so in general it is outperformed by BSML-S.

The crucial quantity used in computing various selection criteria is the GDF (1.48). In the BSML procedures, the GDF is estimated using the same Monte Carlo method discussed in Section 1.7.1. As mentioned earlier, GDF is a good measure of model complexity, but is computationally costly to estimate when an analytical formula is not known. The BSML-C procedure combines the non-null libraries together, therefore the GDF estimates that need to be computed are $\{\hat{D}(\mathcal{M}_{m+1}), \ldots, \hat{D}(\mathcal{M}_M)\}$, where $\mathcal{M}_k$ is the modeling procedure at the kth step of forward selection. Hence, the Monte Carlo algorithm is called $M - m$ times in the BSML-C procedure. For the BSML-S procedure,

however, because it treats different libraries separately, things become more complicated. Two different types of GDFs are computed. One is the basis selection cost within each library, denoted as $\hat{D}(\mathcal{M}_{k,l})$, for $l = 1, \ldots, L$, where $\mathcal{M}_{k,l}$ is the modeling procedure that selects the basis function $\psi_{l,j_l}^{(k)}$ in the lth non-null library given the model $\mathcal{B}_{k-1}$ and performs the estimation based on $\mathcal{B}_{k-1} \cup \{\psi_{l,j_l}^{(k)}\}$. The other is the average selection cost for each library, denoted as $\hat{D}(\mathcal{M}_k^{(l)})$, where $\mathcal{M}_k^{(l)}$ is the modeling procedure that selects $k-m$ additional basis functions $\{\varphi_{l,j_l}, j = 1, \ldots, k-m\}$ one at a time from library $\mathcal{L}_l$ given the model $\mathcal{B}_m$ and performs the estimation based on $\mathcal{B}_m \cup \big(\bigcup_{j=1}^{k-m}\{\varphi_{l,j_l}\}\big)$. Hence, with the same upper bound M on the total number of basis functions to be selected, the BSML-S procedure calls the Monte Carlo algorithm $2L \cdot (M-m)$ times. If L is large, the BSML-S procedure becomes very computationally demanding. For this reason, application of the BSML procedure mainly focuses on the univariate case only, with small number of libraries.

Here we give the details of the BSML-S procedure. At step k, where $k \in \{m+1, \ldots, M\}$, the procedure first selects a basis function $\psi_{l,j_l}^{(k)}$ for each $l \in \{1, \ldots, L\}$ among the remaining basis functions that have not yet been selected in library $\mathcal{L}_l$, to maximize the reduction in the RSS:

$$\psi_{l,j_l}^{(k)} = \underset{\psi \in \mathcal{L}_l \cap \mathcal{B}_{k-1}^{\mathsf{c}}}{\operatorname{argmax}} \big\{\mathrm{RSS}(\mathcal{B}_{k-1}) - \mathrm{RSS}(\mathcal{B}_{k-1} \cup \{\psi\})\big\}. \tag{1.56}$$

Then among these candidate basis functions $\Psi_k = \left\{\psi_{1,j_1}^{(k)}, \ldots, \psi_{L,j_L}^{(k)}\right\}$, the BSML-S procedure selects the one $\phi_k \in \Psi_k$ to minimize the doubly penalized criterion, i.e.,

$$\phi_k = \underset{\psi_{l,j_l}^{(k)} \in \Psi_k}{\operatorname{argmin}} \ \mathrm{DPC}(\psi_{l,j_l}^{(k)}), \tag{1.57}$$

and

$$\mathrm{DPC}(\psi_{l,j_l}^{(k)}) = \mathrm{RSS}\Big(\mathcal{B}_{k-1} \cup \big\{\psi_{l,j_l}^{(k)}\big\}\Big) + c_1\sigma^2\hat{D}(\mathcal{M}_{k,l}) + c_2\sigma^2\hat{A}_k(\mathcal{L}_l), \tag{1.58}$$

where $\hat{D}(\mathcal{M}_{k,l})$ is the selection cost for $\psi_{l,j_l}^{(k)}$ in the lth library at step k, $\hat{A}_k(\mathcal{L}_l)$ is the estimated IDF for the lth library up to step k, and c_1, c_2 are constants. The BSML-S procedure computes $\hat{A}_k(\mathcal{L}_l)$ using

$$\hat{A}_k(\mathcal{L}_l) = \frac{\hat{D}(\mathcal{M}_k^{(l)}) - m}{k - m}, \tag{1.59}$$

where $\hat{D}(\mathcal{M}_k^{(l)})$ is the average selection cost for the lth library up to step k, as introduced previously. Sklar et al. (2013) suggested $c_1 = \ln 2$ and $c_2 = 2$. See the supplement materials of Sklar et al. (2013) for the estimation of σ^2.

In the elimination step, the covariance inflation criterion (CIC) or the generalized cross-validation (GCV) criterion can be used to select the final model. As shown in (1.53), $\mathrm{MSE}(\mathcal{B}_k) = \mathbb{E}(\mathrm{RSS}(\mathcal{B}_k)) + 2C(\mathcal{M}_k) - n\sigma^2$. Hence, the covariance inflation criterion (CIC) can be defined as follows:

$$\begin{aligned}\mathrm{CIC}(k) &\triangleq \frac{1}{n}\mathrm{RSS}(\mathcal{B}_k) + \frac{2}{n}C(\mathcal{M}_k) \\ &= \frac{1}{n}\mathrm{RSS}(\mathcal{B}_k) + \frac{2}{n}\sigma^2 D(\mathcal{M}_k),\end{aligned} \tag{1.60}$$

which is an unbiased estimate of $\mathrm{MSE}(\mathcal{B}_k) + \sigma^2$. In practice, σ^2 and $D(\mathcal{M}_k)$ are replaced by their estimates. The GCV criterion is defined as

$$\mathrm{GCV}(k) = \frac{\sum_{i=1}^n (y_i - \hat{f}(\boldsymbol{x_i}))^2}{(n - D(\mathcal{M}_k))^2} = \frac{\mathrm{RSS}(\mathcal{B}_k)}{(n - D(\mathcal{M}_k))^2}. \tag{1.61}$$

To estimate the GDF $D(\mathcal{M}_k)$ in each of (1.60) and (1.61), the following approximation

is used:

$$\hat{D}(\mathcal{M}_k) \approx m + \sum_{h=m+1}^{k} \hat{D}(\psi_{l_h,j_{l_h}}^{(h)} | \mathcal{B}_{h-1}) \triangleq m + \sum_{h=m+1}^{k} \max\left\{\hat{D}(\mathcal{M}_{h,l_h}) - (h-1), 1\right\}. \quad (1.62)$$

Putting all things together, the algorithm of the BSML-S procedure is as follows:

The BSML-S procedure

1. *Initialization:* Set $\mathcal{B}_m = \mathcal{L}_0$ and let M be an upper bound on the number of basis functions to be selected (including those in $\mathcal{L}_0$).

2. *Forward selection:* for $k = m+1, \ldots, M$, do

   (a) *Select within each library:* for $l = 1, \ldots, L$, do

      i Select $\psi_{l,j_l}^{(k)} \in \mathcal{L}_l$ according to (1.56).

      ii *Estimate GDFs as needed:* compute $\hat{D}(\mathcal{M}_{k,l})$ and $\hat{D}(\psi_{l,j_l}^{(k)} | \mathcal{B}_{k-1})$.

   (b) *Select between libraries:* select $\phi_k \in \Psi_k$ to minimize (1.58).

   (c) *Update:* $\mathcal{B}_k = \mathcal{B}_{k-1} \cup \{\phi_k\}$.

3. *Elimination:* choose $k^*, m \leq k^* \leq M$, as the minimizer of the CIC in (1.60) or the GCV criterion in (1.61).

In comparison, here is the BSML-C procedure:

The BSML-C procedure

1. *Initialization:* Set $\mathcal{B}_m = \mathcal{L}_0$ and let M be an upper bound on the number of basis functions to be selected (including those in $\mathcal{L}_0$).

2. *Forward selection:* for $k = m+1, \ldots, M$, do

(a) Select ϕ_k among the basis functions not yet been selected in $\mathcal{O}$ to maximize the reduction in RSS:

$$\phi_k = \underset{\psi \in \mathcal{O} \cap \mathcal{B}_{k-1}^{c}}{\operatorname{argmax}} \big\{ \mathrm{RSS}(\mathcal{B}_{k-1}) - \mathrm{RSS}(\mathcal{B}_{k-1} \cup \{\psi\}) \big\}. \tag{1.63}$$

(b) Update $\mathcal{B}_k = \mathcal{B}_{k-1} \cup \{\phi_k\}$.

(c) Estimate $\hat{D}(\mathcal{M}_k)$ and $\hat{C}(\mathcal{M}_k)$.

3. *Elimination:* choose $k^*, m \leq k^* \leq M$, as the minimizer of the CIC in (1.60) or the GCV criterion in (1.61).

Therefore, we can see that in BSML-C all non-null libraries are combined together, so there is no need to select bases and estimate the GDFs separately for each library. This will reduce computation burden, but can sometimes be a disadvantage, especially when the libraries used are very different in nature and need to be treated separately. The performance comparison of BSML-S and BSML-C based on simulations will be presented later in Section 2.1.

Compared to other methods such as HAS and MARS, the better performance of the BSML procedure in terms of the estimation of the true function f, when the true function is spatially inhomogeneous, comes from using the GDF to estimate the costs of selecting additional basis functions, and including multiple libraries so that better basis functions can be selected. However, it still has two drawbacks. First, the search of candidate basis functions at each forward selection step (as given in (1.56) and (1.63)) is greedy. The procedure only allows one candidate basis to be selected after each step. Second, the computation burden of estimating the GDFs is heavy, especially for the BSML-S procedure. In Chapters 3, we develop a new procedure that is less greedy in the forward selection, and at the same time more computationally feasible while still treating bases

from different libraries separately.

Chapter 2

Basis Selection from Multiple Libraries Using Adaptive LASSO

2.1 Adaptive LASSO Basis Selection

As we mentioned in Section 1.4.2, the adaptive LASSO method can be used for variable selection because the L_1 penalty shrinks small coefficients to exact zero. Unlike the regular LASSO method, adaptive LASSO allows different penalties for the coefficients by assigning different weights to their L_1 norms. Consider the same nonparametric regression problem as in (1.22) where $f(\boldsymbol{x}_i)$ is approximated by a linear combination of basis functions, i.e.,

$$f(\boldsymbol{x}_i) = \sum_{j=1}^{k^*} \beta_j \xi_j(\boldsymbol{x}_i), \quad \text{for } \xi_j \in \bigcup_{l=0}^{L} \mathcal{L}_l, \tag{2.1}$$

where $\{\xi_1, \ldots, \xi_m\}$ are the null bases from $\mathcal{L}_0$, and $\{\xi_{m+1}, \ldots, \xi_{k^*}\}$ are selected from $\{\mathcal{L}_1, \ldots, \mathcal{L}_L\}$. Then we may apply the adaptive LASSO method in this basis selection setting, because each basis function needs to be penalized differently.

For the simplicity of discussion, assume that there are 3 libraries: $\mathcal{L}_0$, $\mathcal{L}_1$, and $\mathcal{L}_2$,

where $\mathcal{L}_0$ is the null library and all the basis functions in it are automatically included in every model considered. Here is how we apply the adaptive LASSO method to select basis functions. First, a total of M basis functions are preselected in $\mathcal{O} = \cup_{l=1}^{2}\mathcal{L}_l$ to maximize the reduction of RSS at each step:

$$\psi^{(k)} = \underset{\psi \in \mathcal{O} \cap \mathcal{B}_{k-1}^{\mathsf{c}}}{\operatorname{argmax}} \big\{\mathrm{RSS}(\mathcal{B}_{k-1}) - \mathrm{RSS}(\mathcal{B}_{k-1} \cup \{\psi\})\big\}, \tag{2.2}$$

for $k = m+1, \ldots, M$, where $\mathcal{B}_m = \mathcal{L}_0$, and $\mathcal{B}_k = \mathcal{B}_{k-1} \cup \{\psi^{(k)}\}$. Let $\mathscr{E} = \{\psi^{(m+1)}, \ldots, \psi^{(M)}\}$ be the set of preselected non-null basis functions. Also, let d_k be the reduction of RSS after $\psi^{(k)}$ is added to the model $\mathcal{B}_{k-1}$, i.e.,

$$d_k = \mathrm{RSS}(\mathcal{B}_{k-1}) - \mathrm{RSS}(\mathcal{B}_{k-1} \cup \{\psi^{(k)}\}), \quad k = m+1, \ldots, M. \tag{2.3}$$

Then consider the following objective function:

$$\sum_{i=1}^{n}\left(y_i - \sum_{k=1}^{M}\beta_k\,\psi^{(k)}(\boldsymbol{x}_i)\right)^2 + \sum_{\{j\,:\,\psi^{(j)} \in \mathscr{E} \cap \mathcal{L}_1\}} \lambda_1 d_j^{-\gamma}|\beta_j| + \sum_{\{\ell\,:\,\psi^{(\ell)} \in \mathscr{E} \cap \mathcal{L}_2\}} \lambda_2 d_\ell^{-\gamma}|\beta_\ell|, \tag{2.4}$$

where $\{\psi^{(1)}, \ldots, \psi^{(m)}\} = \mathcal{L}_0$, and $\lambda_1, \lambda_2, \gamma > 0$. Based on (2.4), we can see that the bases in $\mathcal{L}_0$ are not penalized, and the penalties of other preselected bases are inversely related to the reduction of RSS caused by adding each basis. Compare to the objective function in (1.38), here we use the weights $\hat{w}_k = d_k^{-\gamma}$, instead of $\hat{w}_k = |\hat{\beta}_k^{\mathrm{ols}}|^{-\gamma}$ as proposed in Zou (2006), because the former works better in our simulations. Note that, in contrary to the linear models, the goal here is to estimate the nonparametric function f rather than the parameters β_k's. We also tried the SEA-LASSO proposed in Qian and Yang (2013), where the weights in the adaptive LASSO are adjusted by the standard error s_k of $\hat{\beta}_k^{\mathrm{ols}}$, i.e., $\hat{w}_k = (s_k/|\hat{\beta}_k^{\mathrm{ols}}|)^{-\gamma}$, since the basis functions from the same libraries can be highly

correlated. However, the performance is still not as good as that using the reduction of RSS in (2.3) and (2.4). The logic behind assigning weights proportional to the reduction of RSS is analogous to selecting candidate basis functions according to the reduction of RSS. Bases that are important should cause a big decrease in RSS when they are added to the model, so they should be penalized less. On the other hand, the coefficients of bases that make little contribution in reducing the overall RSS will be shrunk to zero. We also penalize non-null libraries differently by assigning tuning parameters to each of them, e.g., λ_1 and λ_2 in (2.4). This controls the selection of basis functions from different libraries when they achieve similar reductions of RSS. The parameters λ_1 and λ_2 are tuned by using the criterion in (2.5).

The upper bound on the total number of candidate models is 2^{M-m} when we consider all the combinations of non-null preselected bases. Let $(\lambda_1, \lambda_2, \gamma)$ be the vector of tuning parameters. Let $\mathscr{M} = \{\mathcal{M}_1, \ldots, \mathcal{M}_q\}$ be the set of all candidate models from minimizing (2.4) for $\lambda_1, \lambda_2, \gamma \in \mathbb{R}_+$. Then the final model is chosen to minimize the Bayesian Information Criterion (Schwarz (1978)):

$$\mathrm{BIC}(\mathcal{M}_j) = n \log \mathrm{RSS}(\mathcal{M}_j) + \log(n)|\mathcal{M}_j|, \tag{2.5}$$

where $\mathrm{RSS}(\mathcal{M}_j)$ is the RSS of the ordinary least squares fit of the model using the basis functions in $\mathcal{M}_j$, and $|\mathcal{M}_j|$ is the number of basis functions contained in $\mathcal{M}_j$. We used the BIC for model selection as suggested in Qian and Yang (2013). In practice, we only consider a finite grid of points for the three tuning parameters. The BIC is then minimized over all the combinations of tuning parameter values.

The whole procedure is summarized below.

The Adaptive LASSO Basis Selection (ALBS) Procedure

1. *Initialization:* Set $\mathcal{B}_m = \mathcal{L}_0$ and let M be an upper bound on the number of basis functions to be preselected (including those in $\mathcal{L}_0$).

2. *Preselection:* for $k = m+1, \ldots, M$, do

   (a) Select $\psi^{(k)}$ according to (2.2). Set $\mathcal{B}_k = \mathcal{B}_{k-1} \cup \{\psi^{(k)}\}$.

   (b) Compute the reduction of RSS d_k defined in (2.3).

3. *Elimination:* For each combination of tuning parameter values $(\lambda_1, \lambda_2, \gamma)$ in a discrete grid, obtain the candidate model by minimizing the adaptive LASSO objective function in (2.4), and add it to the candidate model set $\mathscr{M}$. Choose $\mathcal{M}_{j^*} \in \mathscr{M}$ to minimize the BIC in (2.5).

2.2 Simulations to Compare ALBS, BSML, and HAS

We use similar simulation studies as in Sklar et al. (2013) to compare the performances of the ALBS and the BSML procedures. We use the same six univariate test functions and libraries of basis functions, which are given in Table 2.1 and 2.2. We use the same notations as in Table 1.1 to represent the libraries. The test functions are plotted in Figure 2.1.

We can see that all of the test functions are spatially inhomogeneous. It is therefore difficult to estimate them using only one class of basis functions. In particular, the Sine-Jumps, Heavisine, and Blocks-Curves are discontinuous at certain locations. The design points used for all simulations are the grid points $\{x_i = i/n : i = 1, \ldots, n\}$, where the sample size is $n = 512$. The knots used in the libraries $\mathcal{P}_2, \mathcal{C}_2, \mathcal{T}_0, \mathcal{T}_2$ are grid points

Table 2.1: Test functions used in simulations.

Function Name	True Function $f(x)$	$SD(f)$
Sine-Jumps	$\sin(2\pi x) - 1_{(0.5,1]}(x) + 1_{(0.25,1]}(x)$	1.003
Heavisine	$2.2[4\sin(4\pi x) - \text{sign}(x - 0.3) - \text{sign}(0.72 - x)]$	6.294
Blocks-Curves	$1_{[0.2,0.4)}(x) + \exp[(x + 0.5)^2]1_{[0.4,0.7)}(x) - x^{10}1_{[0.7,1]}(x)$	1.461
LW6	$\sin[2(4x - 2)] + 2\exp[-256(x - 0.5)^2]$	0.839
LW7	$(4x - 2) + 2\exp[-256(x - 0.5)^2]$	1.265
Poly-Sine	$\sin(16\pi x) - 8(x - 0.5)^2 + 8(x - 0.5)^3 1_{(0.5,1]}(x)$	0.840

Table 2.2: Libraries of basis functions used in simulations. Notations of basis libraries can be found in Table 1.1.

Function Name	Basis Libraries		
	$\mathcal{L}_0$	$\mathcal{L}_1$	$\mathcal{L}_2$
Sine-Jumps	$\mathcal{U}_0$	$\mathcal{P}_2$	$\mathcal{T}_0$
Heavisine	$\mathcal{U}_0$	$\mathcal{P}_2$	$\mathcal{T}_0$
Blocks-Curves	$\mathcal{U}_1$	$\mathcal{C}_2$	$\mathcal{T}_0$
LW6	$\mathcal{U}_1$	$\mathcal{C}_2$	$\mathcal{T}_2$
LW7	$\mathcal{U}_1$	$\mathcal{C}_2$	$\mathcal{T}_2$
Poly-Sine	$\mathcal{U}_1$	$\mathcal{F}_{8,25}$	$\mathcal{T}_2$

$\{j/n,\ j = 4, \ldots, n - 3\}$. The response variable is generated according to:

$$y_i = f(x_i) + \varepsilon_i, \quad i = 1, \ldots, n, \quad \varepsilon_i, \ldots, \varepsilon_n \overset{iid}{\sim} N\big(0, \sigma^2\big), \tag{2.6}$$

where σ is chosen such that the signal to noise ratio (SNR) defined as $SD(f)/\sigma$ is fixed at 4 for all six test examples, and $SD(f)$ is defined as

$$SD(f) = \sqrt{\frac{1}{n-1}\sum_{i=1}^{n}\Big[f(x_i) - \frac{1}{n}\sum_{i=1}^{n} f(x_i)\Big]^2}. \tag{2.7}$$

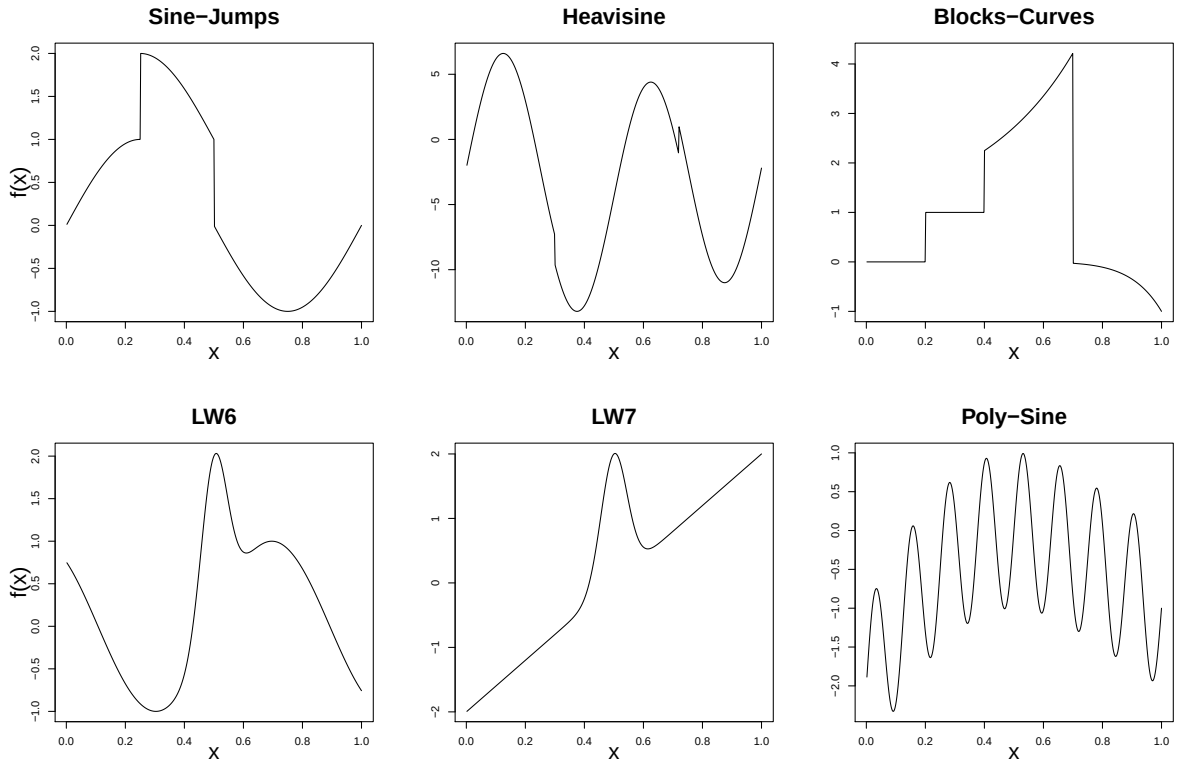


Figure 2.1: Six test functions used in simulations.

We used 100 test sets to compare the performances of four procedures: BSML-S, BSML-C, HAS, and ALBS. Both the HAS and the BSML-C procedures use the combined library $\mathcal{L}_1 \cup \mathcal{L}_2$ in the forward selection, the difference is that BSML-C estimates the GDF by the perturbation method at each step, while HAS fixes the IDF of each selected basis function at 1.2. Although ALBS also carries out the preselection with the combined library, at the following elimination stage it treats the bases from $\mathcal{L}_1$ and $\mathcal{L}_2$ differently by imposing different penalties on them. To make a fair comparison between HAS and ALBS, we used the BIC criterion for the model elimination in the HAS procedure. The performance of all procedures is measured by the mean squared error (MSE):

$$\text{MSE}(\hat{f}) = \frac{1}{n}\sum_{i=1}^{n}\left[\hat{f}(x_i) - f(x_i)\right]^2. \tag{2.8}$$

We used the `R` package `glmnet` (Friedman et al. (2010)) to obtain the adaptive LASSO estimators in the ALBS procedure. We specified the values each tuning parameter can take as follows:

$$\gamma \in \{e^i : i = -5, -4.5, -4, \ldots, 2, 2.5, 3\}, \tag{2.9}$$

$$\frac{\lambda_2}{\lambda_1} \in \{0.1, 0.2, \ldots, 0.8, 0.9, 1, 1.5, 2, \ldots, 4, 4.5, 5\}, \tag{2.10}$$

and λ_1 values are grid points decided automatically inside the `R` function `glmnet`, with the default `nlambda` = 100. In the default setting of `glmnet`, the tuning parameter sequence `lambda` is linear on the log scale from `lambda.max` down to `lambda.min`, where `lambda.max` is the smallest value for `lambda` such that all the coefficients are zero, and by default `lambda.min` = 0.0001× `lambda.max`. See the `glmnet` vignette (Hastie and Qian (2014)) for more details. Therefore, the λ_1 values determined inside the `glmnet` function are different for the six test functions, because for each example the `lambda.max` is different.

For each combination of γ, λ_1, and λ_2, function `glmnet` returns a candidate model, which is then added into the candidate model set $\mathscr{M}$. For the other three procedures HAS, BSML-C, and BSML-S, the `R` package `bsml` (Wu et al. (2012)) was used. For all procedures, the maximal number of bases M was fixed at 30. Figure 2.2 shows the boxplots of the MSEs of 100 test sets using each procedure.

We can see from Figure 2.2 that ALBS had similar performance as HAS for the Blocks-Curves, Heavisine, LW7, and Poly-Sine examples. Although ALBS and HAS preselects exactly the same M basis functions, the set of candidate models considered in ALBS is usually larger than that in HAS. This is because ALBS also penalizes bases differently based on the libraries they are from, so the candidate models returned from the adaptive LASSO algorithm include additional models besides $\mathcal{B}_m, \ldots, \mathcal{B}_M$. However, this

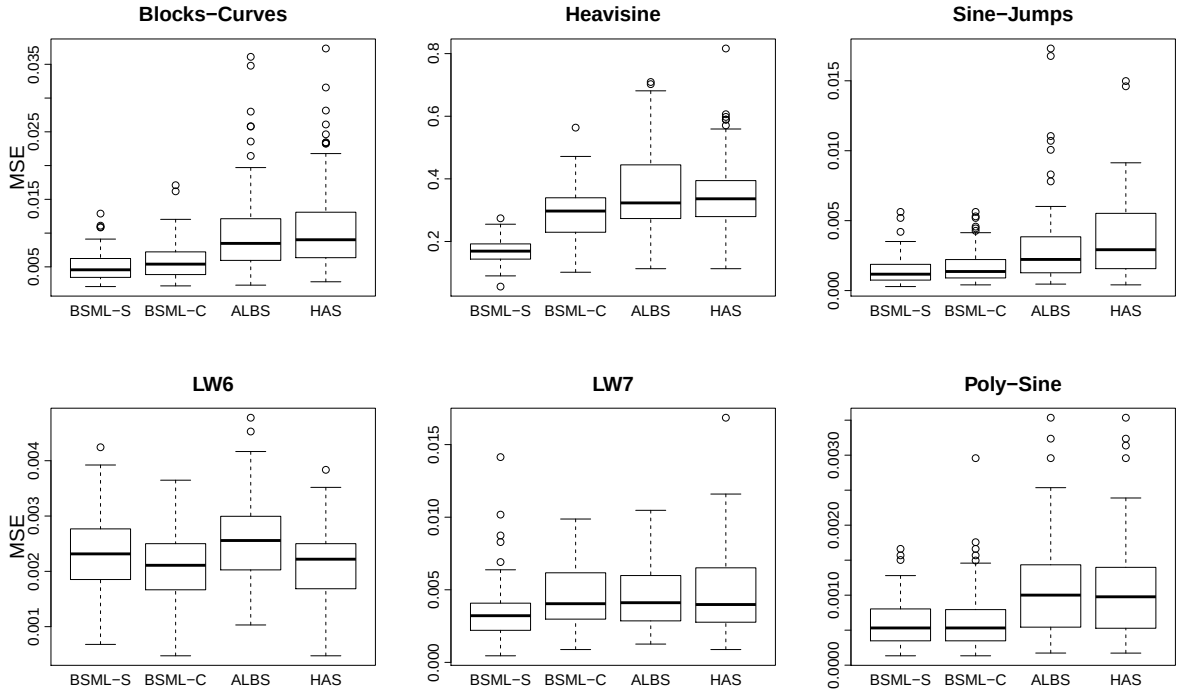


Figure 2.2: Boxplots of the MSEs of 100 test sets using BSML-S, BSML-C, ALBS, and HAS.

did not help the overall performance of the ALBS procedure according to our simulations. ALBS performed worse than the BSML-C procedure (in most cases), especially for the Block-Curves and Heavisine examples. This makes sense because ALBS does not use IDF in the model selection, so in general it tends to select more basis functions than necessary. For example, for the Block-Curves and Heavisine examples, on average BSML-C selected 7 and 12 bases respectively, while ALBS selected 9 and 16 bases respectively. Not surprisingly, the BSML-S procedure had the best overall performance, because it treats the bases from $\mathcal{L}_1$ and $\mathcal{L}_2$ differently, both in the forward selection and in the model elimination with adaptively estimated GDFs.

2.3 Adaptive LASSO Basis Selection with Estimated IDF

As we found in the last section, the ALBS procedure did not perform as well as the BSML procedure. One of the reasons is that ALBS does not use IDF in the model selection, so that it can overfit the data by selecting too many basis functions. To overcome this drawback, we propose a slightly different version of the ALBS procedure by incorporating the adaptively estimated IDF for each library.

Now in the preselection, the same perturbation method mentioned in Section 1.7.1 is used to estimate the IDF of each preselected basis. Let $\hat{D}(\mathcal{M}_k)$ be the estimate of GDF for model $\mathcal{B}_k$. We fixed the IDF of the null bases to be 1, so that $\hat{D}(\mathcal{M}_k) = k$ for $k = 1, \ldots, m$. Let $\Psi_l = \mathscr{E} \cap \mathcal{L}_l$ for $l = 1, 2$. Then the IDF of library $\mathcal{L}_l$ is estimated using:

$$\widehat{\mathrm{IDF}}(\mathcal{L}_l) = \max\left(\frac{1}{|\Psi_l|}\sum_{\{j\,:\,\psi^{(j)}\,\in\,\Psi_l\}}\left(\hat{D}(\mathcal{M}_j) - \hat{D}(\mathcal{M}_{j-1})\right),\ 1\right). \tag{2.11}$$

In the model elimination stage, it uses the following modified BIC to select the final model:

$$\mathrm{BIC}(\mathcal{M}_j) = n\log \mathrm{RSS}(\mathcal{M}_j) + \log(n)\left(m + |\mathcal{M}_j \cap \mathcal{L}_1|\,\widehat{\mathrm{IDF}}(\mathcal{L}_1) + |\mathcal{M}_j \cap \mathcal{L}_2|\,\widehat{\mathrm{IDF}}(\mathcal{L}_2)\right). \tag{2.12}$$

The new procedure is referred as the ALBS-2, which is summarized below.

The ALBS-2 Procedure

1. *Initialization:* Set $\mathcal{B}_m = \mathcal{L}_0$ and let M be an upper bound on the number of basis functions to be preselected (including those in $\mathcal{L}_0$). Set $\hat{D}(\mathcal{M}_k) = k$ for $k = 1, \ldots, m$.

2. *Preselection:* for $k = m+1, \ldots, M$, do

   (a) Select $\psi^{(k)}$ according to (2.2).

   (b) Compute the reduction of RSS d_k defined in (2.3).

   (c) Estimate $\hat{D}(\mathcal{M}_k)$ using the perturbation method.

3. *Elimination:* For each combination of tuning parameter values $(\lambda_1, \lambda_2, \gamma)$ in a discrete grid, obtain the candidate model by minimizing the adaptive LASSO objective function in (2.4), add it to the candidate model set $\mathscr{M}$. Estimate the IDF of each non-null library using (2.11). Choose $\mathcal{M}_{j^*} \in \mathscr{M}$ to minimize the BIC in (2.12).

2.4 Simulations to Compare ALBS-2, BSML, and HAS

We did a similar simulation study as in Section 2.2 to see if ALBS-2 performed better than ALBS. Same 100 test sets were used. The number of perturbations T in the Monte Carlo algorithm was fixed at 50, and the perturbation standard deviation was set at $\tau = 0.5\hat{\sigma}$, where $\hat{\sigma}$ was the Rice estimator of σ (Rice (1984)):

$$\hat{\sigma}_{Rice} = \sqrt{\frac{1}{2(n-1)} \sum_{i=2}^{n} (y_i - y_{i-1})^2}. \tag{2.13}$$

The boxplots of the MSEs are shown in Figure 2.3. With the adaptively estimated IDFs for each library, ALBS-2 performed better than ALBS in the Blocks-Curves, Sine-Jumps, and Poly-Sine examples. Recall that for the Blocks-Curves and Sine-Jumps examples,

$\mathcal{L}_1$ and $\mathcal{L}_2$ contain spline representers and step functions respectively. Thus, for these two examples a bigger IDF for $\mathcal{L}_2$ relatively to $\mathcal{L}_1$ helps ALBS-2 to select fewer step functions when choosing the final model. In the Blocks-Curves example, the average IDFs of $\mathcal{L}_1$ and $\mathcal{L}_2$ were 2.54 and 3.96 respectively. In the Sine-Jumps example, they were 1.88 and 3.98 respectively. Based on the 100 test sets, on average ALBS selected 4.55 bases from $\mathcal{L}_2$ for the Sine-Jumps example, and 6.73 bases from $\mathcal{L}_2$ for the Blocks-Curves example. When ALBS-2 was used, however, these two numbers reduced to 2.03 and 3.08 respectively. The BSML-S procedure was still the best in all examples, but for the Blocks-Curves, Sine-Jumps, and Poly-Sine examples the ALBS-2 procedure performed almost as well. The biggest difference performance-wise between ALBS-2 and BSML-S is with the Heavisine example. The main reason is that for this example, both ALBS-2 and BSML-C combine $\mathcal{L}_1$ (periodic spline representers) and $\mathcal{L}_2$ (step functions) in the preselection process, while the BSML-S compares the cost of selecting bases from $\mathcal{L}_1$ and from $\mathcal{L}_2$ at every step to decide which library to choose from. For the Heavisine example, when BSML-C, ALBS, and ALBS-2 were used, not enough bases from $\mathcal{L}_1$ were preselected, and hence fewer were chosen in the final model. This can be seen from Figure 2.4.

The average CPU time of fitting one test set for each procedure and example used is shown in Table 2.3. Overall, the ALBS-2 procedure does not have clear advantages in terms of both performance and speed when compared to the BSML-C procedure. The reason is that they share the same forward selection process, and only differ in how the elimination is carried out.

Although the ALBS and ALBS-2 procedures we proposed so far are only for the special case when there are two non-null libraries, they can be easily extended to general case of L non-null libraries. However, the potential problem with our procedures is that as the number of libraries grows, so does the number of tuning parameters. It

can be easily seen that the number of combinations of gridded parameter values grows exponentially in the number of libraries. Thus, the computation for the ALBS and ALBS-2 procedures is prohibitive when there are too many libraries, for example in the multivariate nonparametric regression case. Another drawback of the ALBS-2 procedure is the way it estimates the IDFs. It is computationally expensive to have to estimate the IDFs at each preselection step, with repeated perturbations. The BSML procedures share the same drawback, especially BSML-S because it needs to estimate the selection cost for each library as well.

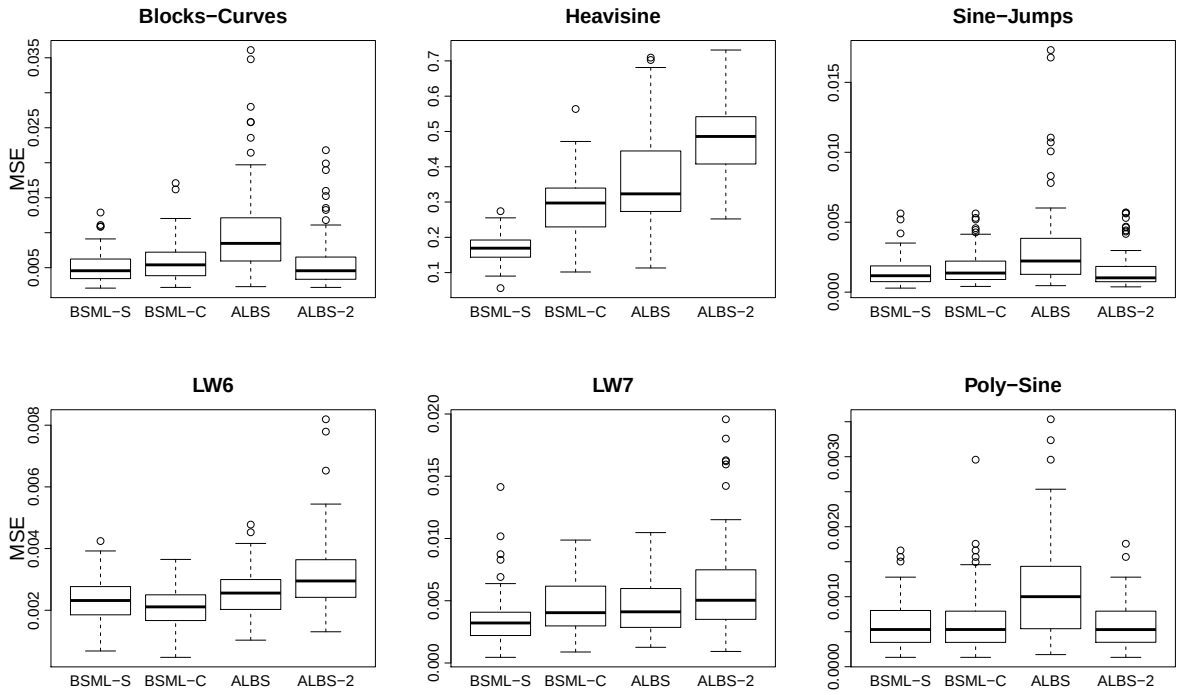


Figure 2.3: Boxplots of the MSEs of 100 test sets using BSML-S, BSML-C, ALBS, and ALBS-2.

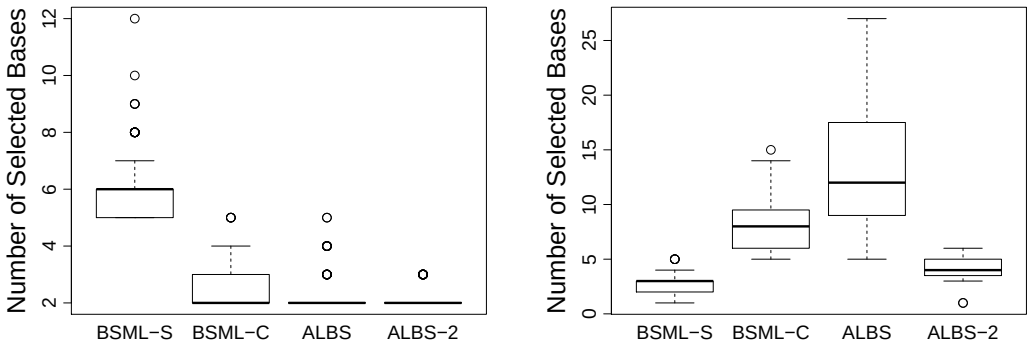


Figure 2.4: Number of selected bases in $\mathcal{L}_1$ (left) and $\mathcal{L}_2$ (right) for the Heavisine example with 100 test sets using BSML-S, BSML-C, ALBS, and ALBS-2.

Table 2.3: Average CPU time (in seconds) based on simulations in Sections 2.2 and 2.4.

	Blocks-Curves	Heavisine	LW6	LW7	Poly-Sine	Sine-Jumps
HAS	3.01	2.92	2.94	2.92	0.77	2.93
ALBS	4.13	7.47	7.63	7.78	4.17	4.35
ALBS-2	7.08	9.62	9.31	9.29	5.59	6.70
BSML-C	8.93	9.29	9.26	9.23	4.30	9.22
BSML-S	51.85	51.33	51.70	51.37	28.81	51.20

Based on the simulations results, there are a few things we have learned so far. First of all, each library needs to be assigned with a unique IDF that depends on the data. Also, in the forward selection process, bases from different libraries need to be considered separately. These findings are the main motivations for the look-ahead procedure we will discuss in the next chapter. Because it is costly to estimate the IDFs, the look-ahead procedure instead treats the IDFs as tuning parameters. Meanwhile, the look-ahead procedure evaluates the candidate bases from each individual library in forward selection, and is less greedy in the searching process.

Chapter 3

Look-Ahead Procedure

3.1 Problems With Greedy Search

As mentioned in Section 1.7.1, one of the disadvantages of the BSML procedure is that the GDFs need to be estimated at each step in the forward selection, which can be computationally expensive. Another disadvantage of the BSML procedure comes from its greediness in adding basis functions. At each forward selection step it performs a greedy search for a single basis function that maximizes the reduction in the residual sum of squares, i.e., it only looks one step ahead. This greedy (short-sighted) nature may lead to the selection of a basis function that is not optimal in the long run.

We propose the look-ahead strategy to help alleviate the greediness in the searching for basis functions. The idea of look-ahead is that when choosing the next moves, instead of only evaluating the outcome obtained in a single step, we look further steps ahead and inspect the performance of some extra potential moves. This approach has been widely studied in the context of tree-search and constructive heuristics. See Frost and Dechter (1995), Bertsekas et al. (1997), and Voss et al. (2005). The look-ahead idea has also been applied in sequential Monte Carlo method (e.g., Zhang and Liu (2002)), and in variable

selection, e.g., Zhang et al. (2007), where multiple look-ahead steps are implemented in the best subset selection.

Our look-ahead procedure is less greedy compared to the BSML procedure because it considers more candidate basis functions when performing forward selection. Assuming that there are L non-null libraries, then each time the procedure updates a current model, it considers not only the best single basis function from each library, but also a second basis function among all the bases that have not yet been selected, to form a pair with the best single basis, yielding a total of at most $2L$ candidate bases. Hence, the search space is much bigger than that of the BSML procedure at each update. This look-ahead idea is important particularly in the multiple libraries setting because now it is possible for two basis functions from different libraries to be selected together as a pair. The optimal pair of basis functions may be different if we choose to select them one step at a time, since in that case the selection of the second basis function is conditioning on the previously selected basis function already in the model. Our look-ahead procedure is a compromise between the most greedy one-step ahead procedure (e.g. BSML) and exhaustive search which is computationally prohibitive.

Although one can also modify the forward selection in the BSML procedure to make it look ahead, the increased search space will make the estimation of the GDFs even more computationally expensive. We will develop a computationally feasible procedure that does not require estimating the IDFs.

The look-ahead procedure mainly consists of two parts. The first is the forward selection part and the second is the elimination part. Let's focus on the forward selection part first.

3.2 Forward Selection In the Look-Ahead Procedure with Fixed IDFs

In the forward selection stage of the look-ahead procedure, the procedure takes in some IDF tuning vectors as input and uses them in the calculation of a criterion for building the candidate models. For now, let us focus on the scenario of building a candidate model given a specific IDF tuning vector.

The BSML procedure selects one basis function at each step in the forward selection. In comparison, the look-ahead procedure can select either one or a pair of basis functions to enter the model each time. Suppose that there are in total $L+1$ libraries, $\{\mathcal{L}_0, \mathcal{L}_1, \ldots, \mathcal{L}_L\}$, where $\mathcal{L}_0$ is the null library with m basis functions that will be included automatically. Let $\mathcal{B}_m$ be the model that includes all the basis functions from the null library, i.e., $\mathcal{B}_m = \mathcal{L}_0$. As the forward selection continues, more and more basis functions are added to $\mathcal{B}_m$, so we obtain a sequence of nested models. The maximum number of basis functions a model can contain is set to be M [1]. Denote $|S|$ as the cardinality of a set S. Denote $\mathcal{B}_-$ as the model that will be updated next during the iterations, and define index k as $k = |\mathcal{B}_-| + 1$. If there is at least one new basis function being added to $\mathcal{B}_-$ after the update, then the number of basis functions in $\mathcal{B}_-$ will either increase by one if one basis is selected, or increase by two if a pair of basis functions is selected. Correspondingly $|\mathcal{B}_-|$ will increase by either one or two at the next iteration.

We start with $\mathcal{B}_- = \mathcal{L}_0$ and $k = m+1$. As long as $k < M$, the look-ahead procedure performs the forward selection as follows. Suppose there are L_- libraries among $\{\mathcal{L}_1, \ldots, \mathcal{L}_L\}$ that contain at least one unselected basis function, denote them as $\mathcal{L}_{(1)}, \ldots, \mathcal{L}_{(L_-)}$, and $\mathcal{O}_- = \cup_{l=1}^{L_-} \mathcal{L}_{(l)}$. The procedure first selects an ordered pair of basis

[1] This upper bound is the user-specified variable `MAXBAS` in the code.

functions

$$\psi_l^- = \underset{\psi \in \mathcal{L}_{(l)} \cap \mathcal{B}_-^{\mathsf{c}}}{\operatorname{argmax}} \big\{\mathrm{RSS}(\mathcal{B}_-) - \mathrm{RSS}(\mathcal{B}_- \cup \{\psi\})\big\}, \tag{3.1}$$

and

$$\psi_{\cdot|l}^- = \underset{\psi \in \mathcal{O}_- \cap \left[\mathcal{B}_- \cup \{\psi_l^-\}\right]^{\mathsf{c}}}{\operatorname{argmax}} \left\{\mathrm{RSS}\Big(\mathcal{B}_- \cup \big\{\psi_l^-\big\}\Big) \; - \; \mathrm{RSS}\Big(\mathcal{B}_- \cup \big\{\psi_l^-, \psi\big\}\Big)\right\}, \tag{3.2}$$

for each $l \in \{1, 2, \ldots, L_-\}$ to form candidates. Let Ψ^- be the collection of all candidate singleton and pairs

$$\Psi^- = \Big\{\big\{\psi_1^-\big\}, \ldots, \big\{\psi_{L_-}^-\big\}, \big\{\psi_1^-, \psi_{\cdot|1}^-\big\}, \ldots, \big\{\psi_{L_-}^-, \psi_{\cdot|L_-}^-\big\}\Big\}, \tag{3.3}$$

where the first L_- candidates in (3.3) are single basis functions from (3.1) and the rest are ordered pairs of basis functions. Here some pairs in Ψ^- could be the same except for order, and these redundant pairs are removed from Ψ^-. Then $|\Psi^-|$ satisfies $L_- + 1 \leq |\Psi^-| \leq 2L_-$. Also, note that for the second basis function in these pairs, i.e., $\psi_{\cdot|l}^-$, it is selected given ψ_l^- in the model, and it can be any non-null basis function that has not yet been selected besides ψ_l^-. This selection scheme is different from the one that selects the second basis functions from each of the L_- libraries, i.e., the scheme that finds in total $\frac{1}{2}L_-(L_- + 1)$ pairs to maximize the reduction of RSS,

$$\big\{\psi_\ell^-, \psi_\kappa^-\big\} = \left\{\psi_a, \psi_b : \underset{\substack{\psi_a \in \mathcal{L}_{(\ell)} \cap \mathcal{B}_-^{\mathsf{c}} \\ \psi_b \in \mathcal{L}_{(\kappa)} \cap \mathcal{B}_-^{\mathsf{c}}}}{\operatorname{argmax}} \left\{\mathrm{RSS}(\mathcal{B}_-) \; - \; \mathrm{RSS}\Big(\mathcal{B}_- \cup \big\{\psi_a, \psi_b\big\}\Big)\right\}\right\}, \tag{3.4}$$

for $1 \leq \ell, \kappa \leq L_-$. Then because $|\mathcal{B}_-| = k - 1$, $\Big|\mathcal{O}_- \cap \big\{\mathcal{B}_- \cup \{\psi_l^-\}\big\}^{\mathsf{c}}\Big| = |\mathcal{O}_-| - k + m$, so for the scheme in (3.2), it searches through $|\mathcal{O}_-| - k + m$ basis functions L_- times, or $(|\mathcal{O}_-| - k + m)L_-$ in total. In comparison, for the scheme in (3.4), it searches through

$\binom{|\mathcal{O}_-|-k+m+1}{2} = \frac{1}{2}\big(|\mathcal{O}_-| - k + m + 1\big)\big(|\mathcal{O}_-| - k + m\big)$ basis functions. Usually, we have $L_- \ll \frac{1}{2}\big(|\mathcal{O}_-| - k + m + 1\big)$, so that the first scheme requires much less computational time. Of course, the trade-off of using (3.2) is that the pairs of basis functions selected may not be the optimal compared to these selected using (3.4). However, based on our simulation experience, the improvement in terms of MSE is too small to justify the extra computations required by the all pairwise search scheme.

After the look-ahead procedure obtains the candidate set Ψ^- in (3.3), we then select $\Psi^-_{j^*} \in \Psi^-$ to maximize the reduction of the BIC. Alternatively, other selection criteria such as the AIC (Akaike (1974)) could also be used. Let $\mathcal{B}^-_j = \mathcal{B}_- \cup \Psi^-_j$ be the jth candidate model for $j = 1, 2, \ldots, |\Psi^-|$, where Ψ^-_j is the jth element of Ψ^-:

$$\Psi^-_j = \begin{cases} \big\{\psi^-_j\big\}, & \text{if } 1 \le j \le L_-, \\ \big\{\psi^-_{j-L_-}, \psi^-_{\cdot|j-L_-}\big\}, & \text{if } L_- + 1 \le j \le |\Psi^-|. \end{cases} \tag{3.5}$$

The BIC is defined as:

$$\mathrm{BIC}\big(\mathcal{B}^-_j\big) = n \log\Big(\mathrm{RSS}\big(\mathcal{B}^-_j\big)\Big) + \log(n)\mathrm{DF}\big(\mathcal{B}^-_j\big), \tag{3.6}$$

where $\mathrm{DF}\big(\mathcal{B}^-_j\big)$ is the degrees of freedom for model $\mathcal{B}^-_j$ defined as follows:

$$\begin{aligned} \mathrm{DF}\big(\mathcal{B}^-_j\big) = \mathrm{DF}(\mathcal{B}_-) &+ \mathrm{IDF}\big(\mathcal{L}(\psi^-_j)\big) I(1 \le j \le L_-) \\ &+ \Big[\mathrm{IDF}\big(\mathcal{L}(\psi^-_{j-L_-})\big) + \mathrm{IDF}\big(\mathcal{L}(\psi^-_{\cdot|j-L_-})\big)\Big] I(L_- + 1 \le j \le |\Psi^-|), \end{aligned} \tag{3.7}$$

where $\mathrm{DF}(\mathcal{B}_-)$ is the degrees of freedom of model $\mathcal{B}_-$, $I(\cdot)$ is the indicator function, $\mathcal{L}\big(\psi^-_{\cdot|j-L_-}\big)$ represents the library that the basis function $\psi^-_{\cdot|j-L_-}$ is in. $\mathrm{DF}(\mathcal{B}_-)$ can be

computed using

$$\mathrm{DF}(\mathcal{B}_-) = \mathrm{DF}(\mathcal{B}_m) + \sum_{i=m+1}^{k-1} \mathrm{IDF}\big(\mathcal{L}(\xi_i)\big) = m + \sum_{i=m+1}^{k-1} \mathrm{IDF}\big(\mathcal{L}(\xi_i)\big), \tag{3.8}$$

where $\xi_{m+1}, \cdots, \xi_{k-1}$ are the non-null bases contained in $\mathcal{B}_-$. Note that $\mathrm{DF}(\mathcal{B}_m) = m$ because the IDFs of all the basis functions in the null library $\mathcal{L}_0$ are fixed at 1.

Let $\Psi^-_{j^*}$ be the set in Ψ^- such that the model $\mathcal{B}_- \cup \Psi^-_{j^*}$ has the lowest BIC, i.e., the index j^* is chosen such that

$$j^* = \underset{1 \le j \le |\Psi^-|}{\operatorname{argmin}} \mathrm{BIC}\big(\mathcal{B}^-_j\big). \tag{3.9}$$

Alternatively, if AIC is used, (3.6) and (3.9) change into

$$\mathrm{AIC}\big(\mathcal{B}^-_j\big) = n \log\Big(\mathrm{RSS}\big(\mathcal{B}^-_j\big)\Big) + 2\mathrm{DF}\big(\mathcal{B}^-_j\big) \tag{3.10}$$

$$j^* = \underset{1 \le j \le |\Psi^-|}{\operatorname{argmin}} \mathrm{AIC}\big(\mathcal{B}^-_j\big). \tag{3.11}$$

Note that if $\mathrm{BIC}(\mathcal{B}^-_{j^*})$ is greater or equal to $\mathrm{BIC}(\mathcal{B}_-)$, then this means adding the basis set $\Psi^-_{j^*}$ to the current model $\mathcal{B}_-$ will likely cause overfitting, so nothing should be added to $\mathcal{B}_-$ instead, and the forward selection for the current model should be stopped. Otherwise the procedure generates either one or two new candidate models depending on how many bases are selected (corresponding to $\Psi^-_{j^*}$). If $j^* \le L_-$, then the current model is replaced with $\mathcal{B}_- \cup \big\{\psi^-_{j^*}\big\}$. If $j^* > L_-$, then the current model is replaced with $\mathcal{B}_- \cup \big\{\psi^-_{j^*-L_-}, \psi^-_{\bullet|j^*-L_-}\big\}$. To start the next iteration, $\mathcal{B}_-$ is set to be the updated model, and $k = |\mathcal{B}_-| + 1$. As long as $k < M$, the same forward selection routine in (3.1), (3.2), (3.5), (3.6), (3.7), and (3.9) is repeated with the new $\mathcal{B}_-$ and k. We can see that different from the BSML procedure, the look-ahead procedure is able to stop the

forward selection before the Mth step if the BIC starts to increase. This will help to avoid the overfitting problem when building each model in the forward selection stage and to reduce computation. Another difference of the look-ahead procedure is that it can select two basis functions simultaneously when performing the forward selection, as we have mentioned earlier.

If $k = M$, which means there are already $M - 1$ basis functions in the current model $\mathcal{B}_-$, the look-ahead procedure will select at most one more basis function so that after this step the number of basis functions in the model will not exceed M. The selection criterion at this step is:

$$\psi_l^- = \underset{\psi \in \mathcal{L}_{(l)} \cap \mathcal{B}_-^c}{\operatorname{argmax}} \big\{\mathrm{RSS}(\mathcal{B}_-) - \mathrm{RSS}(\mathcal{B}_- \cup \{\psi\})\big\}, \quad \text{for } 1 \le l \le L_-, \tag{3.12}$$

$$l^* = \underset{1 \le l \le L_-}{\operatorname{argmin}} \mathrm{BIC}\Big(\mathcal{B}_- \cup \big\{\psi_l^-\big\}\Big). \tag{3.13}$$

If $\mathrm{BIC}\Big(\mathcal{B}_- \cup \big\{\psi_{l^*}^-\big\}\Big) < \mathrm{BIC}(\mathcal{B}_-)$, then the procedure replaces the current model with $\mathcal{B}_- \cup \big\{\psi_{l^*}^-\big\}$.

If $k > M$, the forward selection for the current model will stop.

3.3 Forward Selection Via Householder Transformation

To find the basis function in (3.1), (3.2), and (3.12), we need a fast algorithm to compute the reduction of RSS when one or two new basis functions are added to a model. We are going to use the Householder transformation as in BSML and HAS for selection of a single basis function, and we will extend the computational methods to the case of adding a pair of new basis functions.

Seber and Lee (2003) provides a good discussion of how to use the Householder QR decomposition to calculate the regression coefficients and RSS efficiently. We will summarize some results here.

Consider the linear regression model:

$$\boldsymbol{y} = \boldsymbol{X}\boldsymbol{\beta} + \boldsymbol{\varepsilon}, \tag{3.14}$$

where $\boldsymbol{y} = (y_1, \ldots, y_n)'$ is the $n \times 1$ response vector, $\boldsymbol{X}$ is an $n \times k$ design matrix of full column rank [2], $\boldsymbol{\beta} = (\beta_1, \ldots, \beta_k)'$ is the $k \times 1$ coefficients vector, and $\boldsymbol{\varepsilon} = (\varepsilon_1, \ldots, \varepsilon_n)'$ is the $n \times 1$ vector of random errors with zero mean and constant variance. The least squares estimate $\hat{\boldsymbol{\beta}}$ can be obtained by solving the normal equation:

$$\boldsymbol{X}'\boldsymbol{X}\hat{\boldsymbol{\beta}} = \boldsymbol{X}'\boldsymbol{y}. \tag{3.15}$$

After $\hat{\boldsymbol{\beta}}$ is obtained, the RSS can be calculated as:

$$\text{RSS} = \boldsymbol{e}'\boldsymbol{e}, \tag{3.16}$$

where $\boldsymbol{e} = \boldsymbol{y} - \boldsymbol{X}\hat{\boldsymbol{\beta}}$ are the residuals. It is usually computationally expensive to compute $(\boldsymbol{X}'\boldsymbol{X})^{-1}$ directly when k is large. The idea is to transform $\boldsymbol{X}$ into an upper triangular matrix, so that $\hat{\boldsymbol{\beta}}$ in (3.15) can be solved by back substitution.

3.3.1 The Householder QR Decomposition

The Householder transformation can be used to transform the columns of $\boldsymbol{X}$ so that the elements in the lower triangular will be all zeros. We now describe the Householder transformation. Suppose we have an $n \times 1$ vector $\boldsymbol{x} = (x_1, x_2, \ldots, x_n)'$, and we want to

[2] In our setting, the columns of $\boldsymbol{X}$ represent basis functions evaluated at design points.

rotate it so that the 2nd through nth elements become zero in the rotated space without changing the Euclidean norm of $\boldsymbol{x}$. Hence, in the rotated space $\widetilde{\boldsymbol{x}} = (\|\boldsymbol{x}\|, 0, \ldots, 0)' = \|\boldsymbol{x}\|\boldsymbol{e}_1$, where $\widetilde{\boldsymbol{x}}$ denotes the vector after the rotation, $\|\cdot\|$ is the Euclidean norm, and $\boldsymbol{e}_1 = (1, 0, \ldots, 0)'$ is the n-dimensional vector with only the first element nonzero. Define a Householder vector

$$\boldsymbol{u} = \boldsymbol{x} - \|\boldsymbol{x}\|\boldsymbol{e}_1 = \big(x_1 - \|\boldsymbol{x}\|,\ x_2,\ \ldots,\ x_n\big)', \tag{3.17}$$

then $\boldsymbol{x} - \boldsymbol{u}$ will be the vector after the rotation. As mentioned in Seber and Lee (2003), when calculating the first element, $x_1 - \|\boldsymbol{x}\|$, of $\boldsymbol{u}$, severe cancellation would occur if x_1 is positive and large compared with the other elements of $\boldsymbol{x}$. Thus, we use a slightly different version of the Householder vector:

$$\begin{aligned}
\boldsymbol{u}^* &= \boldsymbol{x} + \operatorname{sign}(x_1)\|\boldsymbol{x}\|\boldsymbol{e}_1 \\
&= \begin{cases} \big(x_1 + \|\boldsymbol{x}\|,\ x_2,\ \ldots,\ x_n\big)', & \text{if } x_1 \geq 0, \\ \big(x_1 - \|\boldsymbol{x}\|,\ x_2,\ \ldots,\ x_n\big)', & \text{if } x_1 < 0, \end{cases}
\end{aligned}$$

where $\operatorname{sign}(x)$ is 1 if $x \geq 0$, and -1 if $x < 0$. Then the transformed vector becomes:

$$\begin{aligned}
\widetilde{\boldsymbol{x}} &= \boldsymbol{x} - \boldsymbol{u}^* \\
&= -\operatorname{sign}(x_1)\|\boldsymbol{x}\|\boldsymbol{e}_1 \\
&= \begin{cases} \big(-\|\boldsymbol{x}\|, 0, \ldots, 0\big)', & \text{if } x_1 \geq 0, \\ \big(\|\boldsymbol{x}\|, 0, \ldots, 0\big)', & \text{if } x_1 < 0. \end{cases}
\end{aligned} \tag{3.18}$$

To make the Householder matrix defined later to have a simpler form, we scale $\boldsymbol{u}^*$ so

that it has Euclidean norm 1:

$$\begin{aligned}\boldsymbol{w} &= \frac{\boldsymbol{u}^*}{\|\boldsymbol{u}^*\|} \\ &= \frac{\boldsymbol{x} + \operatorname{sign}(x_1)\|\boldsymbol{x}\|\boldsymbol{e}_1}{\sqrt{2\|\boldsymbol{x}\|\big(\|\boldsymbol{x}\| + |x_1|\big)}} \qquad (3.19)\\ &= \begin{cases} \Big(\sqrt{\dfrac{\|\boldsymbol{x}\| + x_1}{2\|\boldsymbol{x}\|}}\ ,\ \dfrac{x_2}{\sqrt{2\|\boldsymbol{x}\|(\|\boldsymbol{x}\| + x_1)}},\ \ldots,\ \dfrac{x_n}{\sqrt{2\|\boldsymbol{x}\|(\|\boldsymbol{x}\| + x_1)}}\Big)', & \text{if } x_1 \geq 0, \\ \Big(-\sqrt{\dfrac{\|\boldsymbol{x}\| - x_1}{2\|\boldsymbol{x}\|}},\ \dfrac{x_2}{\sqrt{2\|\boldsymbol{x}\|(\|\boldsymbol{x}\| - x_1)}},\ \ldots,\ \dfrac{x_n}{\sqrt{2\|\boldsymbol{x}\|(\|\boldsymbol{x}\| - x_1)}}\Big)', & \text{if } x_1 < 0. \end{cases} \qquad (3.20)\end{aligned}$$

Using $\boldsymbol{x} - \|\boldsymbol{u}^*\|\boldsymbol{w}$ to obtain the vector $\widetilde{\boldsymbol{x}}$ is equivalent to define the following Householder transformation matrix $\boldsymbol{H}$, and then applying to $\boldsymbol{x}$:

$$\boldsymbol{H} = \boldsymbol{I} - 2\boldsymbol{w}\boldsymbol{w}'. \qquad (3.21)$$

It is easy to see that $\boldsymbol{H}\boldsymbol{x} = \widetilde{\boldsymbol{x}}$ by using the facts $\|\boldsymbol{w}\| = 1$, $\widetilde{\boldsymbol{x}} = \boldsymbol{x} - \|\boldsymbol{u}^*\|\boldsymbol{w}$, and $\|\boldsymbol{x}\| = \|\widetilde{\boldsymbol{x}}\|$:

$$\begin{aligned} 2\boldsymbol{w}'\boldsymbol{x} &= \frac{2}{\|\boldsymbol{u}^*\|}\big(\boldsymbol{x}' - \widetilde{\boldsymbol{x}}'\big)\boldsymbol{x} \\ &= \frac{2}{\|\boldsymbol{u}^*\|}\big(\boldsymbol{x}'\boldsymbol{x} - \widetilde{\boldsymbol{x}}'\boldsymbol{x}\big) \\ &= \frac{1}{\|\boldsymbol{u}^*\|}\big(2\|\boldsymbol{x}\|^2 - 2\widetilde{\boldsymbol{x}}'\boldsymbol{x}\big) \\ &= \frac{1}{\|\boldsymbol{u}^*\|}\big(\|\boldsymbol{x}\|^2 - 2\widetilde{\boldsymbol{x}}'\boldsymbol{x} + \|\widetilde{\boldsymbol{x}}\|^2\big) \\ &= \frac{1}{\|\boldsymbol{u}^*\|}\|\boldsymbol{x} - \widetilde{\boldsymbol{x}}\|^2 \\ &= \frac{1}{\|\boldsymbol{u}^*\|}\Big\|\|\boldsymbol{u}^*\|\boldsymbol{w}\Big\|^2 \\ &= \|\boldsymbol{u}^*\|\|\boldsymbol{w}\|^2 \end{aligned}$$

$$= \|\boldsymbol{u}^*\|.$$

Consequently,

$$\begin{aligned}\boldsymbol{H}\boldsymbol{x} &= \boldsymbol{x} - 2\boldsymbol{w}\boldsymbol{w}'\boldsymbol{x} \\ &= \boldsymbol{x} - \boldsymbol{w}\|\boldsymbol{u}^*\| \\ &= \widetilde{\boldsymbol{x}}.\end{aligned}$$

The Householder matrix $\boldsymbol{H}$ has some nice properties. It is obvious the $\boldsymbol{H}$ is symmetric and orthogonal:

$$\begin{aligned}\boldsymbol{H}'\boldsymbol{H} &= \boldsymbol{H}^2 \\ &= (\boldsymbol{I} - 2\boldsymbol{w}\boldsymbol{w}')(\boldsymbol{I} - 2\boldsymbol{w}\boldsymbol{w}') \\ &= \boldsymbol{I} - 4\boldsymbol{w}\boldsymbol{w}' + 4\boldsymbol{w}(\boldsymbol{w}'\boldsymbol{w})\boldsymbol{w}' \\ &= \boldsymbol{I} - 4\boldsymbol{w}\boldsymbol{w}' + 4\|\boldsymbol{w}\|^2\boldsymbol{w}\boldsymbol{w}' \\ &= \boldsymbol{I}.\end{aligned}$$

We can utilize the Householder transformation to perform QR factorization. A rectangular matrix $\boldsymbol{A} \in \mathbb{R}^{n\times k}$ with $n \geq k$ can be factored into a product of an orthogonal matrix $\boldsymbol{Q} \in \mathbb{R}^{n\times n}$ and an upper triangular matrix $\boldsymbol{U} \in \mathbb{R}^{n\times k}$:

$$\boldsymbol{A} = \boldsymbol{Q}\boldsymbol{U}. \tag{3.22}$$

This factorization is referred to as the QR factorization. Note that if A has full column rank, then by Theorem 5.2.2 in Golub and Loan (2013) the "diagonal" elements of $\boldsymbol{U}$ are nonzero, i.e., $\boldsymbol{U}(j,j) \neq 0$ for $j = 1, \ldots, k$. We assume A to have full column rank from here on. To transform $\boldsymbol{A}$ into $\boldsymbol{U}$, the Householder transformation can be applied to "zero-out" certain elements for the columns in $\boldsymbol{A}$. Here is how it works. Denote the

columns of the rank-k $n \times k$ matrix $\boldsymbol{A}$ as: $\boldsymbol{A}(:,1), \boldsymbol{A}(:,2), \ldots, \boldsymbol{A}(:,k)$. First, choose $\boldsymbol{H}_1$ to zero out all but the first element of $\boldsymbol{A}(:,1)$. Following (3.19) and (3.21), let

$$\boldsymbol{H}_1 = \boldsymbol{I}_n - 2\boldsymbol{w}_1\boldsymbol{w}_1', \quad \text{with } \boldsymbol{w}_1 = \frac{\boldsymbol{A}(:,1) + \text{sign}(\boldsymbol{A}(1,1))\|\boldsymbol{A}(:,1)\|\boldsymbol{e}_1}{\sqrt{2\|\boldsymbol{A}(:,1)\|\big(\|\boldsymbol{A}(:,1)\| + |\boldsymbol{A}(1,1)|\big)}}, \tag{3.23}$$

where $\boldsymbol{A}(1,1)$ is the element in the first row and first column of $\boldsymbol{A}$. Premultiplying $\boldsymbol{A}$ by $\boldsymbol{H}_1$ gives

$$\boldsymbol{H}_1\boldsymbol{A} = \begin{pmatrix} r_{11} & r_{12} & \cdots & r_{1k} \\ 0 & & & \\ \vdots & & \boldsymbol{A}_1 & \\ 0 & & & \end{pmatrix},$$

with $r_{11} = -\text{sign}(\boldsymbol{A}(1,1))\|\boldsymbol{A}(:,1)\|$, and $r_{1j} = \boldsymbol{H}_1(1,:)\boldsymbol{A}(:,j)$ for $2 \le j \le k$. Next consider a matrix $\boldsymbol{H}_2$ of the form

$$\boldsymbol{H}_2 = \begin{pmatrix} 1 & 0 & \cdots & 0 \\ 0 & & & \\ \vdots & & \boldsymbol{K}_2 & \\ 0 & & & \end{pmatrix},$$

where $\boldsymbol{K}_2$ is an $(n-1) \times (n-1)$ Householder matrix chosen to zero out all but the first element of the first column of the submatrix $\boldsymbol{A}_1$, i.e.,

$$\boldsymbol{K}_2 = \boldsymbol{I}_{n-1} - 2\boldsymbol{w}_2\boldsymbol{w}_2', \quad \text{with } \boldsymbol{w}_2 = \frac{\boldsymbol{A}_1(:,1) + \text{sign}(\boldsymbol{A}_1(1,1))\|\boldsymbol{A}_1(:,1)\|\boldsymbol{e}_1}{\sqrt{2\|\boldsymbol{A}_1(:,1)\|\big(\|\boldsymbol{A}_1(:,1)\| + |\boldsymbol{A}_1(1,1)|\big)}}, \tag{3.24}$$

We get

$$\boldsymbol{H}_2\boldsymbol{H}_1\boldsymbol{A} = \begin{pmatrix} r_{11} & r_{12} & r_{13} & \cdots & r_{1k} \\ 0 & r_{22} & r_{23} & \cdots & r_{2k} \\ 0 & 0 & & & \\ \vdots & \vdots & & \boldsymbol{A}_2 & \\ 0 & 0 & & & \end{pmatrix},$$

with $r_{22} = -\text{sign}(\boldsymbol{A}_1(1,1))\|\boldsymbol{A}_1(:,1)\|$, and $r_{2j} = \boldsymbol{K}_2(1,:)\boldsymbol{A}_1(:,j-1)$ for $3 \leq j \leq k$. If we continue analogous steps, the last premultiplication matrix is:

$$\boldsymbol{H}_k = \begin{pmatrix} 1 & 0 & \cdots & 0 \\ 0 & \ddots & \ddots & \vdots \\ \vdots & \ddots & 1 & 0 \\ 0 & \cdots & 0 & \boldsymbol{K}_k \end{pmatrix},$$

where $\boldsymbol{K}_k$ is an $(n-k+1) \times (n-k+1)$ Householder matrix chosen so that the last $n-k$ elements in the kth column of $\boldsymbol{H}_k \cdots \boldsymbol{H}_1\boldsymbol{A}$ are all zeros. Therefore, we have:

$$\boldsymbol{H}_k\boldsymbol{H}_{k-1}\cdots\boldsymbol{H}_1\boldsymbol{A} = \begin{pmatrix} \boldsymbol{R} \\ \hline \boldsymbol{0} \end{pmatrix} = \left(\begin{array}{ccccc} r_{11} & r_{12} & r_{13} & \cdots & r_{1k} \\ 0 & r_{22} & r_{23} & \cdots & r_{2k} \\ \vdots & 0 & r_{33} & \cdots & r_{3k} \\ \vdots & \vdots & 0 & \ddots & \vdots \\ \vdots & \vdots & \vdots & \ddots & r_{kk} \\ \hline \vdots & \vdots & \vdots & \vdots & 0 \\ \vdots & \vdots & \vdots & \vdots & \vdots \\ 0 & 0 & 0 & 0 & 0 \end{array}\right).$$

Here $\boldsymbol{R}$ is an upper-triangular matrix. We define $\boldsymbol{H}_j$ as

$$\boldsymbol{H}_j = \begin{cases} \boldsymbol{H}_1, & \text{if } j = 1, \\ \begin{pmatrix} \boldsymbol{I}_{j-1} & \boldsymbol{0} \\ \boldsymbol{0} & \boldsymbol{K}_j \end{pmatrix}, & \text{if } 2 \le j \le k. \end{cases} \tag{3.25}$$

Note that $\boldsymbol{H}_j^2 = \boldsymbol{H}_j' \boldsymbol{H}_j = \boldsymbol{I}_n$ for $j = 1, \ldots, k$, because of the symmetric and orthogonal properties of the Householder matrices. If we define $\boldsymbol{Q} = (\boldsymbol{H}_k \boldsymbol{H}_{k-1} \cdots \boldsymbol{H}_1)' = \boldsymbol{H}_1 \boldsymbol{H}_2 \cdots \boldsymbol{H}_k$, then we have the Householder QR decomposition:

$$\boldsymbol{A} = \boldsymbol{Q} \begin{pmatrix} \boldsymbol{R} \\ \boldsymbol{0} \end{pmatrix}, \tag{3.26}$$

since

$$\begin{aligned} \boldsymbol{A} &= \boldsymbol{I}_n \boldsymbol{A} \\ &= \boldsymbol{H}_1^2 \boldsymbol{A} \\ &= \boldsymbol{H}_1 \boldsymbol{I}_n \boldsymbol{H}_1 \boldsymbol{A} \\ &= \boldsymbol{H}_1 \boldsymbol{H}_2 \boldsymbol{H}_2 \boldsymbol{H}_1 \boldsymbol{A} \\ &= \boldsymbol{H}_1 \boldsymbol{H}_2 \boldsymbol{I}_n \boldsymbol{H}_2 \boldsymbol{H}_1 \boldsymbol{A} \\ &= \boldsymbol{H}_1 \boldsymbol{H}_2 \boldsymbol{H}_3 \boldsymbol{H}_3 \boldsymbol{H}_2 \boldsymbol{H}_1 \boldsymbol{A} \\ &= \cdots \\ &= (\boldsymbol{H}_1 \cdots \boldsymbol{H}_k)(\boldsymbol{H}_k \cdots \boldsymbol{H}_1 \boldsymbol{A}) \\ &= \boldsymbol{Q} \begin{pmatrix} \boldsymbol{R} \\ \boldsymbol{0} \end{pmatrix}. \end{aligned}$$

Note that $\boldsymbol{Q}$ is orthogonal, because it is a product of orthogonal matrices $\boldsymbol{H}_j$'s. Further-

more, $\boldsymbol{Q}$ can be written as $\boldsymbol{Q} = \big(\boldsymbol{Q}_k, \boldsymbol{Q}_{n-k}\big)$, where $\boldsymbol{Q}_k$ is $n \times k$ and $\boldsymbol{Q}_{n-k}$ is $n \times (n-k)$, so that

$$\boldsymbol{A} = \big(\boldsymbol{Q}_k, \boldsymbol{Q}_{n-k}\big) \begin{pmatrix} \boldsymbol{R} \\ \boldsymbol{0} \end{pmatrix} = \boldsymbol{Q}_k \boldsymbol{R}.$$

For the linear regression setting as in (3.14), let $\boldsymbol{A}$ be the $n \times k$ design matrix $\boldsymbol{X}$ with full column rank, i.e., $\boldsymbol{A} = \boldsymbol{X}$, then by (3.26) for the augmented matrix $\big(\boldsymbol{X}, \boldsymbol{y}\big)$ we have:

$$\big(\boldsymbol{X}, \boldsymbol{y}\big) = \big(\boldsymbol{Q}_k, \boldsymbol{Q}_{n-k}\big) \begin{pmatrix} \boldsymbol{R} & \boldsymbol{r}_1 \\ \boldsymbol{0} & \boldsymbol{r}_2 \end{pmatrix}, \tag{3.27}$$

where $\boldsymbol{R}$ is a $k \times k$ upper triangular matrix, $\boldsymbol{r}_1$ is a $k \times 1$ vector, and $\boldsymbol{r}_2$ is a $(n-k) \times 1$ vector, with

$$\begin{pmatrix} \boldsymbol{r}_1 \\ \boldsymbol{r}_2 \end{pmatrix} = \boldsymbol{Q}'\boldsymbol{y} = \boldsymbol{H}_k \boldsymbol{H}_{k-1} \cdots \boldsymbol{H}_1 \boldsymbol{y},$$

i.e., $(\boldsymbol{r}_1', \boldsymbol{r}_2')'$ is obtained by premultiplying the transformation matrices $\boldsymbol{H}_1, \ldots, \boldsymbol{H}_k$ to $\boldsymbol{y}$. Multiplying out and equating both sides of (3.27) gives

$$\boldsymbol{X} = \boldsymbol{Q}_k \boldsymbol{R}, \tag{3.28}$$

$$\boldsymbol{y} = \boldsymbol{Q}_k \boldsymbol{r}_1 + \boldsymbol{Q}_{n-k} \boldsymbol{r}_2. \tag{3.29}$$

Thus, the solution $\hat{\boldsymbol{\beta}}$ to the normal equation (3.15) is

$$\begin{aligned} \hat{\boldsymbol{\beta}} &= (\boldsymbol{X}'\boldsymbol{X})^{-1}\boldsymbol{X}'\boldsymbol{y} \\ &= (\boldsymbol{R}'\boldsymbol{Q}_k'\boldsymbol{Q}_k\boldsymbol{R})^{-1}\boldsymbol{R}'\boldsymbol{Q}_k(\boldsymbol{Q}_k\boldsymbol{r}_1 + \boldsymbol{Q}_{n-k}\boldsymbol{r}_2) \\ &= (\boldsymbol{R}'\boldsymbol{R})^{-1}\boldsymbol{R}'\boldsymbol{r}_1 \\ &= \boldsymbol{R}^{-1}\boldsymbol{r}_1, \end{aligned} \tag{3.30}$$

where we used the facts that $\boldsymbol{Q}_k'\boldsymbol{Q}_k = \boldsymbol{I}_k$ and $\boldsymbol{Q}_k'\boldsymbol{Q}_{n-k} = \boldsymbol{0}$ because $\boldsymbol{Q} = \big(\boldsymbol{Q}_k, \boldsymbol{Q}_{n-k}\big)$ is

an orthogonal matrix. Equation (3.30) means $\hat{\boldsymbol{\beta}}$ is the solution of $\boldsymbol{R}\hat{\boldsymbol{\beta}} = \boldsymbol{r}_1$. Because $\boldsymbol{R}$ is upper triangular, $\hat{\boldsymbol{\beta}}$ can be solved by back-substitution fairly easily:

$$\begin{cases} \hat{\beta}_k = \boldsymbol{r}_1(k)/\boldsymbol{R}(k,k), \\ \hat{\beta}_j = \left[\boldsymbol{r}_1(j) - \sum_{j'=j+1}^{k} \boldsymbol{R}(j,j')\hat{\beta}_{j'}\right]/\boldsymbol{R}(j,j), \quad \text{for } \ j = k-1, \ldots, 2, 1. \end{cases} \tag{3.31}$$

In addition, the residuals $\boldsymbol{e}$ and the RSS in (3.16) can be written as:

$$\begin{aligned} \boldsymbol{e} &= \boldsymbol{y} - \boldsymbol{X}\hat{\boldsymbol{\beta}} \\ &= \boldsymbol{Q}_k\boldsymbol{r}_1 + \boldsymbol{Q}_{n-k}\boldsymbol{r}_2 - \boldsymbol{Q}_k\boldsymbol{R}\boldsymbol{R}^{-1}\boldsymbol{r}_1 \\ &= \boldsymbol{Q}_{n-k}\boldsymbol{r}_2 \\ &= (\boldsymbol{Q}_k, \boldsymbol{Q}_{n-k}) \begin{pmatrix} \boldsymbol{0} \\ \boldsymbol{r}_2 \end{pmatrix} \\ &= \boldsymbol{Q} \begin{pmatrix} \boldsymbol{0} \\ \boldsymbol{r}_2 \end{pmatrix}, \end{aligned}$$

so that

$$\begin{aligned} \text{RSS} &= \|\boldsymbol{e}\|^2 \\ &= (\boldsymbol{0}', \boldsymbol{r}_2')\boldsymbol{Q}'\boldsymbol{Q}(\boldsymbol{0}', \boldsymbol{r}_2')' \\ &= (\boldsymbol{0}', \boldsymbol{r}_2')(\boldsymbol{0}', \boldsymbol{r}_2')' \\ &= \boldsymbol{r}_2'\boldsymbol{r}_2. \end{aligned} \tag{3.32}$$

3.3.2 Adding A Single Basis Function Using Householder QR Decomposition

Using the Householder QR decomposition, we can easily compute the reduction of RSS as in (3.1) and select a single candidate basis function from each library. Suppose for the current model $\mathcal{B}_-$, the basis functions (including the m bases in $\mathcal{L}_0$) are $\mathcal{B}_- = \{\phi_1, \ldots, \phi_{k-1}\}$ where $k > m$. Let $\boldsymbol{X}_{k-1} = (\boldsymbol{x}_1, \ldots, \boldsymbol{x}_{k-1})$ be the design matrix corresponding to the basis functions in $\mathcal{B}_-$, where the column vector $\boldsymbol{x}_j$ is the basis function ϕ_j evaluated at the design points for each $j \in \{1, \ldots, k-1\}$. Let $\boldsymbol{H}_j$ be the transformation matrix as defined in (3.25), and

$$\boldsymbol{Q} = \boldsymbol{H}_1 \boldsymbol{H}_2 \cdots \boldsymbol{H}_{k-1}, \tag{3.33}$$

then similar to (3.27) we have

$$\left(\boldsymbol{X}_{k-1}, \boldsymbol{y}\right) = \boldsymbol{Q} \begin{pmatrix} \boldsymbol{R} & \boldsymbol{r}_1 \\ 0 & r_2 \\ \boldsymbol{0} & \boldsymbol{r}_3 \end{pmatrix}, \tag{3.34}$$

where $(\boldsymbol{r}_1', r_2, \boldsymbol{r}_3')' = \boldsymbol{Q}'\boldsymbol{y}$. Note that here $\boldsymbol{r}_1$ and $\boldsymbol{r}_3$ are vectors with length $k-1$ and $n-k$ respectively, while r_2 is a scalar. Now we want to add a new basis function ψ to $\mathcal{B}_-$, which corresponds to adding a new column $\boldsymbol{x}$ to $\boldsymbol{X}_{k-1}$. Let $\boldsymbol{q} = \boldsymbol{Q}'\boldsymbol{x} = (\boldsymbol{q}_1', q_2, \boldsymbol{q}_3')'$ where $\boldsymbol{q}_1', q_2$, and $\boldsymbol{q}_3'$ have the same dimensions as $\boldsymbol{r}_1', r_2$, and $\boldsymbol{r}_3'$ respectively. Then we have

$$\left(\boldsymbol{X}_{k-1}, \boldsymbol{x}, \boldsymbol{y}\right) = \boldsymbol{Q} \begin{pmatrix} \boldsymbol{R} & \boldsymbol{q}_1 & \boldsymbol{r}_1 \\ 0 & q_2 & r_2 \\ \boldsymbol{0} & \boldsymbol{q}_3 & \boldsymbol{r}_3 \end{pmatrix}. \tag{3.35}$$

To compute the reduction of RSS caused by adding this new basis function, we can apply a Householder transformation to zero out $\boldsymbol{q}_3$. Let $\boldsymbol{K}$ be the Householder matrix for $\boldsymbol{c} = (q_2, \boldsymbol{q}_3')'$, and let $\boldsymbol{c}^* = \boldsymbol{K}\boldsymbol{c} = (q_2^*, \boldsymbol{0}')'$. Let $\boldsymbol{d} = (r_2, \boldsymbol{r}_3')'$ and $\boldsymbol{d}^* = \boldsymbol{K}\boldsymbol{d} = (r_2^*, \boldsymbol{r}_3^{*\prime})'$. Then by the result in (3.32), we have

$$\begin{aligned}\mathrm{RSS}(\mathcal{B}_-) = \boldsymbol{d}'\boldsymbol{d} &= \big(\boldsymbol{K}'\boldsymbol{d}^*\big)'\big(\boldsymbol{K}'\boldsymbol{d}^*\big) = (\boldsymbol{d}^*)'\boldsymbol{K}\boldsymbol{K}'\boldsymbol{d}^* = (\boldsymbol{d}^*)'\boldsymbol{d}^* \\ &= (r_2^*)^2 + (\boldsymbol{r}_3^*)'(\boldsymbol{r}_3^*) \\ &= (r_2^*)^2 + \mathrm{RSS}\big(\mathcal{B}_- \cup \{\psi\}\big). \end{aligned} \tag{3.36}$$

Therefore, the reduction of RSS in (3.1) is:

$$\Delta\mathrm{RSS} = \mathrm{RSS}(\mathcal{B}_-) - \mathrm{RSS}\big(\mathcal{B}_- \cup \{\psi\}\big) = (r_2^*)^2. \tag{3.37}$$

However, it is not computationally efficient to use (3.37) directly to compute the reduction of RSS when ψ is added to $\mathcal{B}_-$. The reason is that in order to find ψ_l^- in (3.1), we need to compute ΔRSS for each of the individual basis functions $\psi \in \mathcal{L}_{(l)} \cap \mathcal{B}_-^{\mathsf{c}}$, but the Householder matrix for each of them is different, so if we use (3.37) directly we need to construct the Householder transformation $\big|\mathcal{L}_{(l)} \cap \mathcal{B}_-^{\mathsf{c}}\big|$ number of times, for each $l \in \{1, 2, \ldots, L_-\}$.

There is a more convenient way to compute ΔRSS without applying the Householder transformation for ψ explicitly. Denote $\boldsymbol{K}(:,1)$ as $\mathbf{k}$, and $\boldsymbol{K}(:, 2:(n-k+1))$ as $\mathbf{K}_r$, then we have:

$$(\boldsymbol{c}, \boldsymbol{d}) = \big(\mathbf{k}, \mathbf{K}_r\big) \begin{pmatrix} q_2^* & r_2^* \\ \boldsymbol{0} & \boldsymbol{r}_3^* \end{pmatrix}. \tag{3.38}$$

Multiplying out and equating both sides of (3.38) gives

$$\boldsymbol{c} = q_2^*\mathbf{k},$$

$$\boldsymbol{d} = r_2^*\mathbf{k} + \mathbf{K}_r\boldsymbol{r}_3^*.$$

Therefore,

$$\begin{aligned}\boldsymbol{c}'\boldsymbol{d} &= q_2^*\mathbf{k}'\big(r_2^*\mathbf{k} + \mathbf{K}_r\boldsymbol{r}_3^*\big)\\ &= q_2^*r_2^* + q_2^*\mathbf{k}'\mathbf{K}_r\boldsymbol{r}_3^*\\ &= q_2^*r_2^*,\end{aligned}$$

$$\begin{aligned}\boldsymbol{c}'\boldsymbol{c} &= \big(q_2^*\mathbf{k}'\big)\big(q_2^*\mathbf{k}\big)\\ &= (q_2^*)^2,\end{aligned}$$

where we used the fact that the Householder matrix $\boldsymbol{K} = \big(\mathbf{k}, \mathbf{K}_r\big)$ is orthogonal. Hence, according to (3.37) we have

$$\Delta\text{RSS} = (r_2^*)^2 = \frac{(\boldsymbol{c}'\boldsymbol{d})^2}{\boldsymbol{c}'\boldsymbol{c}} = \frac{(q_2r_2 + \boldsymbol{q}_3'\boldsymbol{r}_3)^2}{q_2^2 + \boldsymbol{q}_3'\boldsymbol{q}_3}. \tag{3.39}$$

If we use (3.39) to compute ΔRSS, then there is no need to apply any Householder transformation for the basis functions that have not yet been selected, i.e., $\psi \in \mathcal{L}_{(l)} \cap \mathcal{B}_-^{\mathsf{c}}$, in finding ψ_l^- for each $l \in \{1, 2, \ldots, L_-\}$. Therefore, for each $l \in \{1, 2, \ldots, L_-\}$, to find ψ_l^- in $\mathcal{L}_{(l)} \cap \mathcal{B}_-^{\mathsf{c}}$ that maximizes the reduction of RSS, first the Householder transformation based on the basis functions already in $\mathcal{B}_-$, namely $\{\phi_1, \ldots, \phi_{k-1}\}$, is used to obtain the quantities $q_2, \boldsymbol{q}_3, r_2$, and $\boldsymbol{r}_3$. Then ΔRSS in (3.39) is computed for all unselected basis functions in $\mathcal{L}_{(l)} \cap \mathcal{B}_-^{\mathsf{c}}$, and the one with the largest reduction of RSS is chosen to be candidate basis function ψ_l^-. Suppose ϕ_k is the basis function selected among all the candidates at this iteration. At the next iteration, the Householder transformation for ϕ_k is applied to obtain the transformation matrix $\boldsymbol{H}_k$. The design matrix becomes

$\boldsymbol{X}_k = (\boldsymbol{X}_{k-1}, \boldsymbol{x}_k)$, where $\boldsymbol{x}_k$ is ϕ_k evaluated at the design points. The matrix $\boldsymbol{Q}$ at this iteration becomes $\boldsymbol{Q} = \boldsymbol{H}_1\boldsymbol{H}_2\cdots\boldsymbol{H}_k$. ΔRSS in (3.39) can then be computed similarly with the updated quantities $q_2, \boldsymbol{q}_3, r_2$, and $\boldsymbol{r}_3$, and used to select $\left\{\psi_1^-, \ldots, \psi_{L_-}^-\right\}$ at this iteration.

3.3.3 Adding A Pair of Basis Functions Using Householder QR Decomposition

We use the same idea as in the previous section to compute the reduction of RSS when a pair of basis functions are added to the model $\mathcal{B}_-$. Let $\{\psi_1, \psi_2\}$ be any pair of new basis functions, and let $\boldsymbol{x}$ and $\boldsymbol{z}$ be ψ_1 and ψ_2 evaluated at the design points respectively. Using the same definition of $\boldsymbol{Q}$ in (3.33), then similar to (3.34), we have

$$\left(\boldsymbol{X}_{k-1}, \boldsymbol{y}\right) = \boldsymbol{Q}\begin{pmatrix} \boldsymbol{R} & \boldsymbol{r}_1 \\ 0 & r_2 \\ 0 & r_3 \\ \boldsymbol{0} & \boldsymbol{r}_4 \end{pmatrix}, \tag{3.40}$$

where $(\boldsymbol{r}_1', r_2, r_3, \boldsymbol{r}_4')' = \boldsymbol{Q}'\boldsymbol{y}$, the column vectors $\boldsymbol{r}_1$ and $\boldsymbol{r}_4$ have length $k-1$ and $n-k-1$ respectively, and both r_2 and r_3 are scalars. Let $\boldsymbol{p} = \boldsymbol{Q}'\boldsymbol{x} = (\boldsymbol{p}_1', p_2, p_3, \boldsymbol{p}_4')'$ and $\boldsymbol{q} = \boldsymbol{Q}'\boldsymbol{z} = (\boldsymbol{q}_1', q_2, q_3, \boldsymbol{q}_4')'$ where $\boldsymbol{p}_1$ and $\boldsymbol{q}_1$ are column vectors with length $k-1$, and $\boldsymbol{p}_4$ and $\boldsymbol{q}_4$ are column vectors with length $n-k-1$, then similar to (3.35) we have

$$\left(\boldsymbol{X}_{k-1}, \boldsymbol{x}, \boldsymbol{z}, \boldsymbol{y}\right) = \boldsymbol{Q}\begin{pmatrix} \boldsymbol{R} & \boldsymbol{p}_1 & \boldsymbol{q}_1 & \boldsymbol{r}_1 \\ 0 & p_2 & q_2 & r_2 \\ 0 & p_3 & q_3 & r_3 \\ \boldsymbol{0} & \boldsymbol{p}_4 & \boldsymbol{q}_4 & \boldsymbol{r}_4 \end{pmatrix}. \tag{3.41}$$

Denote $\boldsymbol{b} = (p_2, p_3, \boldsymbol{p}_4')'$, $\boldsymbol{c} = (q_2, q_3, \boldsymbol{q}_4')'$, and $\boldsymbol{d} = (r_2, r_3, \boldsymbol{r}_4')'$. Let $\boldsymbol{K}$ be the Householder matrix for $\boldsymbol{b}$, then we have

$$\boldsymbol{K}(\boldsymbol{b}, \boldsymbol{c}, \boldsymbol{d}) = (\boldsymbol{K}\boldsymbol{b}, \boldsymbol{K}\boldsymbol{c}, \boldsymbol{K}\boldsymbol{d}) = \begin{pmatrix} p_2^* & q_2^* & r_2^* \\ 0 & q_3^* & r_3^* \\ \boldsymbol{0} & \boldsymbol{q}_4^* & \boldsymbol{r}_4^* \end{pmatrix}. \tag{3.42}$$

Denote $\boldsymbol{c}^* = \big(q_3^*, (\boldsymbol{q}_4^*)'\big)'$, and $\boldsymbol{d}^* = \big(r_3^*, (\boldsymbol{r}_4^*)'\big)'$. Let $\boldsymbol{K}^*$ be the Householder matrix for $\boldsymbol{c}^*$, then we have

$$\boldsymbol{K}^*(\boldsymbol{c}^*, \boldsymbol{d}^*) = (\boldsymbol{K}^*\boldsymbol{c}^*, \boldsymbol{K}^*\boldsymbol{d}^*) = \begin{pmatrix} q_3^{**} & r_3^{**} \\ \boldsymbol{0} & \boldsymbol{r}_4^{**} \end{pmatrix}. \tag{3.43}$$

From (3.42), we have

$$\boldsymbol{K}\boldsymbol{d} = \begin{pmatrix} r_2^* \\ \boldsymbol{d}^* \end{pmatrix}, \quad \text{or} \quad \boldsymbol{d} = \boldsymbol{K}' \begin{pmatrix} r_2^* \\ \boldsymbol{d}^* \end{pmatrix}.$$

From (3.43), we have

$$\boldsymbol{K}^*\boldsymbol{d}^* = \begin{pmatrix} r_3^{**} \\ \boldsymbol{r}_4^{**} \end{pmatrix}, \quad \text{or} \quad \boldsymbol{d}^* = (\boldsymbol{K}^*)' \begin{pmatrix} r_3^{**} \\ \boldsymbol{r}_4^{**} \end{pmatrix}.$$

Thus, using the result in (3.32) and also the facts that $\boldsymbol{K}\boldsymbol{K}' = \boldsymbol{I}$ and $\boldsymbol{K}^*(\boldsymbol{K}^*)' = \boldsymbol{I}$, we have

$$\begin{aligned} \mathrm{RSS}\big(\mathcal{B}_-\big) &= \boldsymbol{d}'\boldsymbol{d} \\ &= \Big(r_2^*, (\boldsymbol{d}^*)'\Big)\boldsymbol{K}\boldsymbol{K}' \begin{pmatrix} r_2^* \\ \boldsymbol{d}^* \end{pmatrix} \\ &= (r_2^*)^2 + (\boldsymbol{d}^*)'\boldsymbol{d}^* \\ &= (r_2^*)^2 + \Big(r_3^{**}, (\boldsymbol{r}_4^{**})'\Big)\boldsymbol{K}^*(\boldsymbol{K}^*)' \begin{pmatrix} r_3^{**} \\ \boldsymbol{r}_4^{**} \end{pmatrix} \\ &= (r_2^*)^2 + (r_3^{**})^2 + (\boldsymbol{r}_4^{**})'\boldsymbol{r}_4^{**} \end{aligned}$$

$$= (r_2^*)^2 + (r_3^{**})^2 + \mathrm{RSS}\big(\mathcal{B}_- \cup \{\psi_1, \psi_2\}\big)$$

Therefore, the reduction of RSS by adding $\{\psi_1, \psi_2\}$ to model $\mathcal{B}_-$ is:

$$\Delta\mathrm{RSS} = \mathrm{RSS}\big(\mathcal{B}_-\big) - \mathrm{RSS}\big(\mathcal{B}_- \cup \{\psi_1, \psi_2\}\big) = (r_2^*)^2 + (r_3^{**})^2. \tag{3.44}$$

Same as before, we want to avoid applying the Householder transformations for ψ_1 and ψ_2 explicitly. Denote $\boldsymbol{K}(:,1)$ as $\mathbf{k}_1$, $\boldsymbol{K}(:,2)$ as $\mathbf{k}_2$, and $\boldsymbol{K}(:,3:(n-k+1))$ as $\mathbf{K}_r$, then we have

$$(\boldsymbol{b}, \boldsymbol{c}, \boldsymbol{d}) = (\mathbf{k}_1, \mathbf{k}_2, \mathbf{K}_r) \begin{pmatrix} p_2^* & q_2^* & r_2^* \\ 0 & q_3^* & r_3^* \\ \mathbf{0} & \boldsymbol{q}_4^* & \boldsymbol{r}_4^* \end{pmatrix}, \tag{3.45}$$

Multiplying out and equating both sides of (3.45) gives

$$\begin{cases} \boldsymbol{b} = p_2^* \mathbf{k}_1, & (3.46) \\ \boldsymbol{c} = q_2^* \mathbf{k}_1 + q_3^* \mathbf{k}_2 + \mathbf{K}_r \boldsymbol{q}_4^*, & (3.47) \\ \boldsymbol{d} = r_2^* \mathbf{k}_1 + r_3^* \mathbf{k}_2 + \mathbf{K}_r \boldsymbol{r}_4^*. & (3.48) \end{cases}$$

Therefore,

$$\begin{cases} \boldsymbol{b}'\boldsymbol{b} = (p_2^*)^2, & (3.49) \\ \boldsymbol{b}'\boldsymbol{c} = p_2^* q_2^*, & (3.50) \\ \boldsymbol{b}'\boldsymbol{d} = p_2^* r_2^*, & (3.51) \\ \boldsymbol{c}'\boldsymbol{c} = (q_2^*)^2 + (q_3^*)^2 + (\boldsymbol{q}_4^*)'\boldsymbol{q}_4^* = (q_2^*)^2 + (\boldsymbol{c}^*)'\boldsymbol{c}^*, & (3.52) \\ \boldsymbol{c}'\boldsymbol{d} = q_2^* r_2^* + q_3^* r_3^* + (\boldsymbol{q}_4^*)'\boldsymbol{r}_4^* = q_2^* r_2^* + (\boldsymbol{c}^*)'\boldsymbol{d}^*, & (3.53) \end{cases}$$

where we have used the facts that $(\mathbf{k}_1)'\mathbf{k}_1 = (\mathbf{k}_2)'\mathbf{k}_2 = 1$, $(\mathbf{K}_r)'\mathbf{K}_r = \boldsymbol{I}$, $(\mathbf{k}_1)'\mathbf{k}_2 = 0$, and $(\mathbf{k}_1)'\mathbf{K}_r = (\mathbf{k}_2)'\mathbf{K}_r = \mathbf{0}$ because the Householder matrix $\boldsymbol{K} = (\mathbf{k}_1, \mathbf{k}_2, \mathbf{K}_r)$ is orthogonal.

Similarly, denote $\boldsymbol{K}^*(:,1)$ as $\mathbf{k}^*$, and $\boldsymbol{K}^*(:,2:(n-k))$ as $\mathbf{K}_r^*$, then we have

$$(\boldsymbol{c}^*, \boldsymbol{d}^*) = (\mathbf{k}^*, \mathbf{K}_r^*) \begin{pmatrix} q_3^{**} & r_3^{**} \\ \mathbf{0} & \boldsymbol{r}_4^{**} \end{pmatrix}. \tag{3.54}$$

Multiplying out and equating both sides of (3.54) gives

$$\begin{cases} \boldsymbol{c}^* = q_3^{**}\mathbf{k}^*, & (3.55) \\ \boldsymbol{d}^* = r_3^{**}\mathbf{k}^* + \mathbf{K}_r^*\boldsymbol{r}_4^{**}. & (3.56) \end{cases}$$

Therefore,

$$\begin{cases} (\boldsymbol{c}^*)'\boldsymbol{c}^* = (q_3^{**})^2, & (3.57) \\ (\boldsymbol{c}^*)'\boldsymbol{d}^* = q_3^{**}r_3^{**}, & (3.58) \end{cases}$$

where $(\mathbf{k}^*)'\mathbf{k}^* = 1$, $(\mathbf{K}_r^*)'\mathbf{K}_r^* = \boldsymbol{I}$, and $(\mathbf{k}^*)'\mathbf{K}_r^* = \mathbf{0}$ because of the orthogonality of the Householder matrix $\boldsymbol{K}^* = (\mathbf{k}^*, \mathbf{K}_r^*)$. By (3.49) and (3.51), we have:

$$(r_2^*)^2 = \frac{(\boldsymbol{b}'\boldsymbol{d})^2}{\boldsymbol{b}'\boldsymbol{b}}. \tag{3.59}$$

By (3.49) - (3.51),

$$(q_2^*)^2 = \frac{(\boldsymbol{b}'\boldsymbol{c})^2}{\boldsymbol{b}'\boldsymbol{b}}, \quad q_2^* r_2^* = \frac{(\boldsymbol{b}'\boldsymbol{c})(\boldsymbol{b}'\boldsymbol{d})}{\boldsymbol{b}'\boldsymbol{b}}. \tag{3.60}$$

Thus, using (3.52), (3.53), and (3.60), we have

$$\begin{cases} (\boldsymbol{c}^*)'\boldsymbol{c}^* = \boldsymbol{c}'\boldsymbol{c} - (q_2^*)^2 = \boldsymbol{c}'\boldsymbol{c} - \dfrac{(\boldsymbol{b}'\boldsymbol{c})^2}{\boldsymbol{b}'\boldsymbol{b}}, & (3.61) \\ (\boldsymbol{c}^*)'\boldsymbol{d}^* = \boldsymbol{c}'\boldsymbol{d} - q_2^* r_2^* = \boldsymbol{c}'\boldsymbol{d} - \dfrac{(\boldsymbol{b}'\boldsymbol{c})(\boldsymbol{b}'\boldsymbol{d})}{\boldsymbol{b}'\boldsymbol{b}}. & (3.62) \end{cases}$$

Hence, by (3.57), (3.58), and also the results in (3.61) and (3.62), we have

$$\begin{aligned} r_3^{**} &= \frac{\left((\boldsymbol{c}^*)'\boldsymbol{d}^*\right)^2}{(\boldsymbol{c}^*)'\boldsymbol{c}^*} \\ &= \frac{\left(\boldsymbol{c}'\boldsymbol{d} - \dfrac{(\boldsymbol{b}'\boldsymbol{c})(\boldsymbol{b}'\boldsymbol{d})}{\boldsymbol{b}'\boldsymbol{b}}\right)^2}{\boldsymbol{c}'\boldsymbol{c} - \dfrac{(\boldsymbol{b}'\boldsymbol{c})^2}{\boldsymbol{b}'\boldsymbol{b}}} \\ &= \frac{\Big[(\boldsymbol{b}'\boldsymbol{b})(\boldsymbol{c}'\boldsymbol{d}) - (\boldsymbol{b}'\boldsymbol{c})(\boldsymbol{b}'\boldsymbol{d})\Big]^2}{(\boldsymbol{b}'\boldsymbol{b})\Big[(\boldsymbol{b}'\boldsymbol{b})(\boldsymbol{c}'\boldsymbol{c}) - (\boldsymbol{b}'\boldsymbol{c})^2\Big]}. \end{aligned} \tag{3.63}$$

Therefore, by (3.44), (3.59), and (3.63), we have

$$\begin{aligned} \Delta\text{RSS} &= (r_2^*)^2 + (r_3^{**})^2 \\ &= \frac{(\boldsymbol{b}'\boldsymbol{d})^2}{\boldsymbol{b}'\boldsymbol{b}} + \frac{\Big[(\boldsymbol{b}'\boldsymbol{b})(\boldsymbol{c}'\boldsymbol{d}) - (\boldsymbol{b}'\boldsymbol{c})(\boldsymbol{b}'\boldsymbol{d})\Big]^2}{(\boldsymbol{b}'\boldsymbol{b})\Big[(\boldsymbol{b}'\boldsymbol{b})(\boldsymbol{c}'\boldsymbol{c}) - (\boldsymbol{b}'\boldsymbol{c})^2\Big]} \\ &= \frac{(\boldsymbol{b}'\boldsymbol{d})^2(\boldsymbol{c}'\boldsymbol{c}) + (\boldsymbol{b}'\boldsymbol{b})(\boldsymbol{c}'\boldsymbol{d})^2 - 2(\boldsymbol{c}'\boldsymbol{d})(\boldsymbol{b}'\boldsymbol{c})(\boldsymbol{b}'\boldsymbol{d})}{(\boldsymbol{b}'\boldsymbol{b})(\boldsymbol{c}'\boldsymbol{c}) - (\boldsymbol{b}'\boldsymbol{c})^2}, \end{aligned} \tag{3.64}$$

where $\boldsymbol{b} = (p_2, p_3, \boldsymbol{p}_4')'$, $\boldsymbol{c} = (q_2, q_3, \boldsymbol{q}_4')'$, and $\boldsymbol{d} = (r_2, r_3, \boldsymbol{r}_4')'$ as defined earlier.

Similar to the situation in Section 3.3.2, to add a pair of basis functions to $\mathcal{B}_-$, first the Householder transformation based on the basis functions already in $\mathcal{B}_-$, namely $\{\phi_1, \ldots, \phi_{k-1}\}$, is used to obtain the quantities $\boldsymbol{b}$, $\boldsymbol{c}$, and $\boldsymbol{d}$ for each pair of basis functions that are considered. Then ΔRSS in (3.64) is computed for each pair of basis functions, and the pair leading to the largest reduction of RSS is selected to be candidate pair. In the look-ahead procedure, a candidate pair has the form $\left\{\psi_l^-, \psi_{\cdot|l}^-\right\}$, for $l \in 1, 2, \ldots, L_-$, where $\psi_l^- \in \mathcal{L}_{(l)} \cap \mathcal{B}_-^{\mathsf{c}}$. Suppose the pair $\{\phi_k, \phi_{k+1}\}$ is selected among all the candidate basis functions at this iteration. At the next iteration, the Householder transformations for ϕ_k and ϕ_{k+1} are applied to obtain the transformation matrices $\boldsymbol{H}_k$ and $\boldsymbol{H}_{k+1}$ respectively.

The design matrix becomes $\boldsymbol{X}_{k+1} = (\boldsymbol{X}_{k-1}, \boldsymbol{x}_k, \boldsymbol{x}_{k+1})$, where $\boldsymbol{x}_k$ and $\boldsymbol{x}_{k+1}$ are ϕ_k and ϕ_{k+1} evaluated at the design points respectively. The matrix $\boldsymbol{Q}$ at this iteration becomes $\boldsymbol{Q} = \boldsymbol{H}_1\boldsymbol{H}_2\cdots\boldsymbol{H}_k\boldsymbol{H}_{k+1}$. ΔRSS in (3.64) can then be computed similarly with the updated quantities $\boldsymbol{b}$, $\boldsymbol{c}$, and $\boldsymbol{d}$, and used to select the candidate pairs at this iteration.

3.4 Forward Selection In the Look-Ahead Procedure with IDFs as Tuning Parameters

We have discussed how the look-ahead procedure performs the forward selection with fixed IDFs in Section 3.2. However, as mentioned in Section 1.7.1, the IDFs should be chosen adaptively based on the data. We now treat the IDFs as tuning parameters and discuss how this will change the forward selection procedure.

Suppose there are $L+1$ libraries $\{\mathcal{L}_0, \mathcal{L}_1, \ldots, \mathcal{L}_L\}$ where $\mathcal{L}_0$ is the null library and the rest are non-null libraries. The IDF of $\mathcal{L}_0$ is fixed at 1. The IDFs of the non-null libraries are treated as tuning parameters, which can take any positive values, with IDF = 1 meaning the degrees of freedom is not inflated. Let s_l be the IDF of the l-th library. We consider a user-specified finite number of possible values for s_l and denote the collection of these values as S_l, for $l = 1, \ldots, L$. Define the IDF tuning vector as $\boldsymbol{s} = (s_1, \ldots, s_L)$. Then $\boldsymbol{s}$ is in the IDF space $\mathcal{S} = S_1 \times \cdots \times S_L$, which contains all possible combinations of the user-specified IDF values. Obviously, the size of $\mathcal{S}$, $\prod_{l=1}^{L} |S_l|$, grows exponentially in L.

The difference when multiple IDF tuning vectors (several $\boldsymbol{s} \in \mathcal{S}$) are used is that now when BIC($\mathcal{B}_j^-$) is computed using (3.6), it can attain different minimums depending on the involved IDFs. Hence, there can be multiple choices of $\Psi_{j^*}^-$ according to (3.9), each minimizing BIC($\mathcal{B}_j^-$) given a specific combination of IDFs (i.e., given a specific $\boldsymbol{s} \in \mathcal{S}$).

As a result, at each step the forward selection process can generate multiple new models $\mathcal{B}_- \cup \Psi_{j^*}^-(\boldsymbol{s})$ for $\boldsymbol{s} \in \mathcal{S}$, given the current model $\mathcal{B}_-$.

During the forward selection process, we need to keep the records of both the basis functions contained in each model and the IDF tuning vectors that lead to the selection of these basis functions. Let $\mathcal{D}$ be the collection of all models generated at the current stage in the forward selection. Here we refer to a unique collection of basis functions as a model. Denote the hth model in $\mathcal{D}$ as $\mathcal{A}_h$, and let $\mathcal{S}_h$ be the set of IDF tuning vectors that leads to the selection of the basis functions in $\mathcal{A}_h$. We will call $\mathcal{S}_h$ the IDF set of $\mathcal{A}_h$ from now on. Let $\mathcal{V}_-$ be the set of models in $\mathcal{D}$ that the procedure is updating at the current round [3]. At this round, the procedure updates one model in $\mathcal{V}_-$ at each iteration. Note that $\mathcal{V}_-$ stays the same at each round of iterations. Let $\mathcal{V}_+$ be the set of models will be updated at the next round. Before the current round of iterations begins, $\mathcal{V}_- = \mathcal{V}_+$. Then after each model in $\mathcal{V}_-$ is updated, $\mathcal{V}_+$ is adjusted accordingly. If the procedure decides to stop adding more bases to a set in $\mathcal{V}_-$, that model is removed from $\mathcal{V}_+$. If there are new models generated, they are added to $\mathcal{V}_+$ and $\mathcal{D}$. Once all models in $\mathcal{V}_-$ have been updated, the procedure sets $\mathcal{V}_- = \mathcal{V}_+$ for the next round of iterations. We start with a single model $\mathcal{D} = \mathcal{V}_- = \mathcal{V}_+ = \{\mathcal{A}_1\}$, where $\mathcal{A}_1$ contains the m bases from $\mathcal{L}_0$, and we initialize $\mathcal{S}_1 = \mathcal{S}$. Denote the current model being updated by the forward selection process as $\mathcal{B}_-$, and its IDF set as $\mathcal{S}_-$. They are initialized with $\mathcal{B}_- = \mathcal{A}_1$ and $\mathcal{S}_- = \mathcal{S}_1$.

Suppose $\mathcal{D}$, $\mathcal{V}_-$ contain sets of selected basis functions $\mathcal{A}_1, \ldots, \mathcal{A}_d$, where d is the total number of models generated so far. At the first iteration, $d = 1$. Suppose now the procedure is updating $\mathcal{A}_h$ in $\mathcal{V}_-$, where $\mathcal{A}_h$ contains $k-1$ basis functions. Thus, we have $\mathcal{B}_- = \mathcal{A}_h$, with the IDF set $\mathcal{S}_- = \mathcal{S}_h$. Our procedure then follows the similar forward selection routine in (3.1), (3.2), (3.5), (3.7), and (3.9) for each $\boldsymbol{s} \in \mathcal{S}_-$. Recall that after

[3] See the example at the end of this section for more details about each "round" of iterations.

each update, the procedure can add zero, one, or two basis functions to $\mathcal{B}_-$. For any $\boldsymbol{s} \in \mathcal{S}_-$, if $\text{BIC}_{\boldsymbol{s}}(\mathcal{B}_-)$ is smaller or equal to the BICs of all the candidate models, then no basis will be added to $\mathcal{B}_-$. Define

$$\mathcal{S}_{h,0} = \Big\{ \boldsymbol{s} \in \mathcal{S}_- : \text{BIC}_{\boldsymbol{s}}(\mathcal{B}_-) \leq \min_{1 \leq j \leq |\Psi^-|} \text{BIC}_{\boldsymbol{s}}(\mathcal{B}_j^-) \Big\}, \tag{3.65}$$

where $\mathcal{B}_j^- = \mathcal{B}_- \cup \Psi_j^-$ with Ψ_j^- defined in (3.5). Then there are three possibilities of how $\mathcal{A}_h$ is going to be updated:

(i). If $\mathcal{S}_{h,0} = \mathcal{S}_-$, then the forward selection of $\mathcal{A}_h$ will stop and $\mathcal{A}_h$ is removed from $\mathcal{V}_+$. For $\mathcal{A}_h$, its IDF set is $\mathcal{S}_h = \mathcal{S}_{h,0}$.

(ii). If $\mathcal{S}_{h,0} \subsetneq \mathcal{S}_-$ and $\mathcal{S}_{h,0} \neq \emptyset$, then for any given $\boldsymbol{s} \in \mathcal{S}_{h,0}$ none of the BICs of the candidate models is lower than the BIC of $\mathcal{B}_-$, the forward selection of $\mathcal{A}_h$ will stop and $\mathcal{A}_h$ is removed from $\mathcal{V}_+$. $\mathcal{S}_h$ is updated by setting $\mathcal{S}_h = \mathcal{S}_{h,0}$. For each $\boldsymbol{s} \in \mathcal{S}_- \setminus \mathcal{S}_{h,0}$, the procedure finds the set $\Psi_{j^*(\boldsymbol{s})}^-$ in Ψ^- such that the model $\mathcal{B}_{j^*(\boldsymbol{s})}^- = \mathcal{B}_- \cup \Psi_{j^*(\boldsymbol{s})}^-$ has the smallest BIC. The index $j^*(\boldsymbol{s})$ is obtained using

$$j^*(\boldsymbol{s}) = \underset{1 \leq j \leq |\Psi^-|}{\text{argmin}} \, \text{BIC}_{\boldsymbol{s}}(\mathcal{B}_j^-). \tag{3.66}$$

Note that for $\boldsymbol{s}_1 \neq \boldsymbol{s}_2$ where $\boldsymbol{s}_1, \boldsymbol{s}_2 \in \mathcal{S}_- \setminus \mathcal{S}_{h,0}$, $j^*(\boldsymbol{s}_1)$ and $j^*(\boldsymbol{s}_2)$ could be the same. Denote the unique elements of $\big\{ j^*(\boldsymbol{s}) : \boldsymbol{s} \in \mathcal{S}_- \setminus \mathcal{S}_{h,0} \big\}$ as $j_1^*, \ldots, j_q^*$, then $\Psi_{j_1^*}^-, \ldots, \Psi_{j_q^*}^-$ are the unique sets of basis functions selected at the current iteration. For each $\Psi_{j_i^*}^-$ a new model $\mathcal{A}_{h,i}$ is generated:

$$\mathcal{A}_{h,i} = \begin{cases} \mathcal{B}_- \cup \big\{ \psi_{j_i^*}^- \big\}, & \text{if } j_i^* \leq L_-, \\ \mathcal{B}_- \cup \big\{ \psi_{j_i^* - L_-}^-, \psi_{\cdot | j_i^* - L_-}^- \big\}, & \text{if } j_i^* > L_-, \end{cases} \tag{3.67}$$

for $i = 1, \ldots, q$. The IDF set $\mathcal{S}_{h,i}$ for $\mathcal{A}_{h,i}$ is defined as

$$\mathcal{S}_{h,i} = \Big\{ \boldsymbol{s} : \boldsymbol{s} \in \mathcal{S}_- \setminus \mathcal{S}_{h,0}, \mathrm{BIC}_{\boldsymbol{s}}\big(\mathcal{B}_- \cup \Psi^-_{j_i^*}\big) = \min_{1 \le j \le |\Psi^-|} \mathrm{BIC}_{\boldsymbol{s}}\big(\mathcal{B}_j^-\big) \Big\}, \tag{3.68}$$

for $i = 1, \ldots, q$. To add these new sets to $\mathcal{D}$ and $\mathcal{V}_+$, the procedure generates $\mathcal{A}_{d+i} = \mathcal{A}_{h,i}$ together with $\mathcal{S}_{d+i} = \mathcal{S}_{h,i}$, for $i = 1, \ldots, q$, since there are d sets already in $\mathcal{D}$. $\mathcal{A}_{d+1}, \ldots, \mathcal{A}_{d+q}$ are then added to both $\mathcal{D}$ and $\mathcal{V}_+$.

(iii). If $\mathcal{S}_{h,0} = \emptyset$, then the same new sets $\mathcal{A}_{h,1}, \ldots, \mathcal{A}_{h,q}$ as in (ii) are generated. $\mathcal{A}_h \in \mathcal{D}$ and $\mathcal{A}_h \in \mathcal{V}_+$ are then updated by setting $\mathcal{A}_h = \mathcal{A}_{h,1}$, and its IDF set is updated by setting $\mathcal{S}_h = \mathcal{S}_{h,1}$. To add the rest of the new models to $\mathcal{D}$ and $\mathcal{V}_+$ if $q > 1$, the procedure generates $\mathcal{A}_{d+i-1} = \mathcal{A}_{h,i}$ together with $\mathcal{S}_{d+i-1} = \mathcal{S}_{h,i}$, for $i = 2, \ldots, q$. $\mathcal{A}_{d+2}, \ldots, \mathcal{A}_{d+q}$ are then added to both $\mathcal{D}$ and $\mathcal{V}_+$ if $q > 1$.

At the next iteration, the procedure finds the next model available in $\mathcal{V}_-$ and assign it as $\mathcal{B}_-$. Once the update of the last model in $\mathcal{V}_-$ is finished, the procedure updates $\mathcal{V}_-$ by setting $\mathcal{V}_- = \mathcal{V}_+$. Then the forward selection process goes through the same routine again for each model in $\mathcal{V}_-$. The forward selection for each model in $\mathcal{D}$ and $\mathcal{V}_+$ stops when the BIC stops decreasing, or when the number of basis functions exceeds M. The whole forward selection process stops when both $\mathcal{V}_-$ and $\mathcal{V}_+$ are empty sets.

There are two ways the look-ahead procedure prevents overfitting. First, at most M basis functions are allowed to be selected for each model. Second, to prevent having too many models after the forward selection completes, we can also control the maximum number of models to be returned, denoted as M_d. Thus, we have $|\mathcal{D}| \le M_d$, and this helps to reduce the computation burden, especially when many libraries are used. For a certain iteration, if q new models are generated, but $|\mathcal{D}| + q > M_d$, then only the first $M_d - |\mathcal{D}|$ of them are added to $\mathcal{D}$ and $\mathcal{V}_+$, and the rest are discarded. The maximum

number of models that can be generated in the forward selection is $\prod_{l=1}^{L} |S_l|$. This happens when every IDF tuning vector in $\mathcal{S}$ leads to a unique model. Therefore, when we set $M_d \geq \prod_{l=1}^{L} |S_l|$, there is no limitation on the number of models. We used this latter setting in all of our simulations.

Now let us look at a specific example that illustrates the forward selection process in the look-ahead procedure. Consider the Blocks-Curves function defined in Table 2.1. We use similar simulation settings as in Section 2.2, with sample size $n = 512$, grid design points from $1/n$ to 1, and the response values generated with SNR = 3.

We use the same libraries as in Table 2.1 for this example, where $\mathcal{L}_0 = \mathcal{U}_1$ consists of the constant and linear terms, $\mathcal{L}_1 = \mathcal{C}_2$ consists of cubic spline representers, and $\mathcal{L}_2 = \mathcal{T}_0$ consists of step functions. The knots used in $\mathcal{L}_1$ and $\mathcal{L}_2$ are grid points $\{(j + 3)/n,\ j = 1, \ldots, n - 6\}$, so both $\mathcal{L}_1$ and $\mathcal{L}_2$ contain 506 basis functions. In addition, each model allows at most $M = 30$ basis functions to be selected, including both basis functions from $\mathcal{L}_0$. Denote the IDF of $\mathcal{L}_i$ as s_i, for $i = 1, 2$. s_1 and s_2 can take values in $S_1 = S_2 = \{1, 2, 3, 4, 5, 6\}$, so in total there are $6^2 = 36$ IDF tuning vectors in the IDF space $\mathcal{S} = S_1 \times S_2$. The maximum number of candidate models M_d is set to be 36, so none of the new models that could be generated are discarded in the forward selection. The IDF of each basis function in $\mathcal{L}_0$ is fixed at 1. Let ξ_i be the ith basis function in $\mathcal{L}_0 \cup \mathcal{L}_1 \cup \mathcal{L}_2$, so $\mathcal{L}_0 = \{\xi_1, \xi_2\}$, $\mathcal{L}_1 = \{\xi_3, \ldots, \xi_{508}\}$, and $\mathcal{L}_2 = \{\xi_{509}, \ldots, \xi_{1014}\}$. We now present the details of the steps that the look-ahead procedure took in performing the forward selection based on a single realization of $\boldsymbol{y}$. Note that the basis functions selected can change if a different realization of $\boldsymbol{y}$ were used.

Initialization: Set $\mathcal{D} = \mathcal{V}_- = \mathcal{V}_+ = \{\mathcal{A}_1\}$ and $\mathcal{S}_1 = \mathcal{S}$, where $\mathcal{A}_1 = \{\xi_1, \xi_2\}$.

Example Iterations:

(1). Set $\mathcal{B}_- = \mathcal{A}_1$ and $\mathcal{S}_- = \mathcal{S}_1$. For our $\boldsymbol{y}$, the pair $\{\xi_{863}, \xi_{379}\}$ was selected for all

$\boldsymbol{s} \in \mathcal{S}_-$, so $\mathcal{A}_1$ is updated by setting $\mathcal{A}_1 = \{\xi_1, \xi_2, \xi_{863}, \xi_{379}\}$. $\mathcal{S}_1$ stays the same. After the update, $\mathcal{D} = \mathcal{V}_+ = \mathcal{V}_- = \{\mathcal{A}_1\}$.

(2). Set $\mathcal{B}_- = \mathcal{A}_1$ and $\mathcal{S}_- = \mathcal{S}_1$. Given the IDF set $\mathcal{S}_{1,0} = \{(s_1, s_2) \in \mathcal{S} : s_2 > 3\}$, none of the BICs of the candidate models is lower than $\text{BIC}_{\boldsymbol{s}}(\mathcal{B}_-)$, so $\mathcal{A}_1$ completes forward selection for $\boldsymbol{s} \in \mathcal{S}_{1,0}$, and is removed from $\mathcal{V}_+$. $\mathcal{S}_1$ is updated by setting $\mathcal{S}_1 = \mathcal{S}_{1,0}$.

Given the IDF set $\mathcal{S}_{1,1} = \{(s_1, s_2) \in \mathcal{S} : s_2 \leq 3\}$, the pair $\{\xi_{709}, \xi_{607}\}$ is selected, so a new model $\mathcal{A}_2$ is generated with $\mathcal{A}_2 = \mathcal{B}_- \cup \{\xi_{709}, \xi_{607}\} = \{\xi_1, \xi_2, \xi_{863}, \xi_{379}, \xi_{709}, \xi_{607}\}$. The IDF set for $\mathcal{A}_2$ is $\mathcal{S}_2 = \mathcal{S}_{1,1}$. $\mathcal{A}_2$ is then added to $\mathcal{D}$ and $\mathcal{V}_+$. After the update, $\mathcal{D} = \{\mathcal{A}_1, \mathcal{A}_2\}$ and $\mathcal{V}_+ = \{\mathcal{A}_2\}$. $\mathcal{V}_-$ is updated by setting $\mathcal{V}_- = \{\mathcal{A}_2\}$.

(3). Set $\mathcal{B}_- = \mathcal{A}_2$ and $\mathcal{S}_- = \mathcal{S}_2$. Given the IDF set $\mathcal{S}_{2,1} = \{(s_1, s_2) \in \mathcal{S} : s_1 \leq 3, s_2 \leq 2\}$, the pair $\{\xi_{85}, \xi_{970}\}$ is selected. Given the IDF set $\mathcal{S}_{2,2} = \{(s_1, s_2) \in \mathcal{S} : s_1 \geq 4 \text{ and } s_2 \leq 2, \text{ or, } s_1 \geq 5 \text{ and } s_2 = 3\}$, the single basis $\{\xi_{835}\}$ is selected. Given the IDF set $\mathcal{S}_{2,3} = \{(s_1, s_2) \in \mathcal{S} : s_1 \leq 4, s_2 = 3\}$, the pair $\{\xi_{607}, \xi_{85}\}$ is selected. Thus, $\mathcal{A}_2$ is updated by setting $\mathcal{A}_2 = \mathcal{B}_- \cup \{\xi_{85}, \xi_{970}\} = \{\xi_1, \xi_2, \xi_{863}, \xi_{379}, \xi_{709}, \xi_{607}, \xi_{85}, \xi_{970}\}$, and $\mathcal{S}_2$ is updated by setting $\mathcal{S}_2 = \mathcal{S}_{2,1}$. Two new sets $\mathcal{A}_3$ and $\mathcal{A}_4$ are generated, where $\mathcal{A}_3 = \mathcal{B}_- \cup \{\xi_{835}\} = \{\xi_1, \xi_2, \xi_{863}, \xi_{379}, \xi_{709}, \xi_{607}, \xi_{835}\}$ and $\mathcal{A}_4 = \mathcal{B}_- \cup \{\xi_{607}, \xi_{85}\} = \{\xi_1, \xi_2, \xi_{863}, \xi_{379}, \xi_{709}, \xi_{607}, \xi_{85}\}$. The IDF sets for $\mathcal{A}_3$ and $\mathcal{A}_4$ are $\mathcal{S}_3 = \mathcal{S}_{2,2}$ and $\mathcal{S}_4 = \mathcal{S}_{2,3}$ respectively. $\mathcal{A}_3$ and $\mathcal{A}_4$ are then added to $\mathcal{D}$ and $\mathcal{V}_+$. After the update, $\mathcal{D} = \{\mathcal{A}_1, \mathcal{A}_2, \mathcal{A}_3, \mathcal{A}_4\}$ and $\mathcal{V}_+ = \{\mathcal{A}_2, \mathcal{A}_3, \mathcal{A}_4\}$. $\mathcal{V}_-$ is updated by setting $\mathcal{V}_- = \{\mathcal{A}_2, \mathcal{A}_3, \mathcal{A}_4\}$.

(4). Set $\mathcal{B}_- = \mathcal{A}_2$ and $\mathcal{S}_- = \mathcal{S}_2$. For all $\boldsymbol{s} \in \mathcal{S}_-$, none of the BICs of the candidate models is lower than $\text{BIC}_{\boldsymbol{s}}(\mathcal{B}_-)$, so $\mathcal{A}_2$ completes forward selection, and is re-

moved from $\mathcal{V}_+$. After the update, $\mathcal{D} = \{\mathcal{A}_1, \mathcal{A}_2, \mathcal{A}_3, \mathcal{A}_4\}$, $\mathcal{V}_+ = \{\mathcal{A}_3, \mathcal{A}_4\}$, and $\mathcal{V}_- = \{\mathcal{A}_2, \mathcal{A}_3, \mathcal{A}_4\}$.

(5). Set $\mathcal{B}_- = \mathcal{A}_3$ and $\mathcal{S}_- = \mathcal{S}_3$. Given the IDF set $\mathcal{S}_{3,0} = \{(s_1, s_2) \in \mathcal{S} : s_1 \geq 5, s_2 = 3\}$, none of the BICs of the candidate models is lower than $\text{BIC}_{\boldsymbol{s}}(\mathcal{B}_-)$, so $\mathcal{A}_3$ completes forward selection, and is removed from $\mathcal{V}_+$. $\mathcal{S}_3$ is updated by setting $\mathcal{S}_3 = \mathcal{S}_{3,0}$.

Given the IDF set $\mathcal{S}_{3,1} = \{(s_1, s_2) \in \mathcal{S} : s_1 \geq 4, s_2 = 1\}$, the pair $\{\xi_{875}, \xi_{932}\}$ is selected. Given the IDF set $\mathcal{S}_{3,2} = \{(s_1, s_2) \in \mathcal{S} : s_1 \geq 4, s_2 = 2\}$, the single basis $\{\xi_{875}\}$ is selected. Thus, two new models $\mathcal{A}_5$ and $\mathcal{A}_6$ are generated with $\mathcal{A}_5 = \mathcal{B}_- \cup \{\xi_{875}, \xi_{932}\} = \{\xi_1, \xi_2, \xi_{863}, \xi_{379}, \xi_{709}, \xi_{607}, \xi_{835}, \xi_{875}, \xi_{932}\}$ and $\mathcal{A}_6 = \mathcal{B}_- \cup \{\xi_{875}\} = \{\xi_1, \xi_2, \xi_{863}, \xi_{379}, \xi_{709}, \xi_{607}, \xi_{835}, \xi_{875}\}$. Their IDF sets are $\mathcal{S}_5 = \mathcal{S}_{3,1}$, and $\mathcal{S}_6 = \mathcal{S}_{3,2}$. $\mathcal{A}_5$ and $\mathcal{A}_6$ are then added to $\mathcal{D}$ and $\mathcal{V}_+$. After the update, $\mathcal{D} = \{\mathcal{A}_1, \mathcal{A}_2, \mathcal{A}_3, \mathcal{A}_4, \mathcal{A}_5, \mathcal{A}_6\}$, $\mathcal{V}_+ = \{\mathcal{A}_4, \mathcal{A}_5, \mathcal{A}_6\}$, and $\mathcal{V}_- = \{\mathcal{A}_2, \mathcal{A}_3, \mathcal{A}_4\}$.

(6). Set $\mathcal{B}_- = \mathcal{A}_4$ and $\mathcal{S}_- = \mathcal{S}_4$. For all $\boldsymbol{s} \in \mathcal{S}_-$, none of the BICs of the candidate models is lower than $\text{BIC}_{\boldsymbol{s}}(\mathcal{B}_-)$, so $\mathcal{A}_4$ completes forward selection, and is removed from $\mathcal{V}_+$. After the update, $\mathcal{D} = \{\mathcal{A}_1, \mathcal{A}_2, \mathcal{A}_3, \mathcal{A}_4, \mathcal{A}_5, \mathcal{A}_6\}$ and $\mathcal{V}_+ = \{\mathcal{A}_5, \mathcal{A}_6\}$. $\mathcal{V}_-$ is updated by setting $\mathcal{V}_- = \{\mathcal{A}_5, \mathcal{A}_6\}$.

(7). Set $\mathcal{B}_- = \mathcal{A}_5$ and $\mathcal{S}_- = \mathcal{S}_5$. For all $\boldsymbol{s} \in \mathcal{S}_-$, the pair $\{\xi_{984}, \xi_{970}\}$ is selected. Thus, $\mathcal{A}_5$ is updated by setting $\mathcal{A}_5 = \mathcal{B}_- \cup \{\xi_{984}, \xi_{970}\} = \{\xi_1, \xi_2, \xi_{863}, \xi_{379}, \xi_{709}, \xi_{607}, \xi_{835}, \xi_{875}, \xi_{932}, \xi_{984}, \xi_{970}\}$. $\mathcal{S}_5$ stays the same. After the update, $\mathcal{D} = \{\mathcal{A}_1, \mathcal{A}_2, \mathcal{A}_3, \mathcal{A}_4, \mathcal{A}_5, \mathcal{A}_6\}$, $\mathcal{V}_+ = \{\mathcal{A}_5, \mathcal{A}_6\}$, and $\mathcal{V}_- = \{\mathcal{A}_5, \mathcal{A}_6\}$.

(8). Set $\mathcal{B}_- = \mathcal{A}_6$ and $\mathcal{S}_- = \mathcal{S}_6$. For all $\boldsymbol{s} \in \mathcal{S}_-$, none of the BICs of the candidate models is lower than $\text{BIC}_{\boldsymbol{s}}(\mathcal{B}_-)$, so $\mathcal{A}_6$ completes forward selection, and is removed from $\mathcal{V}_+$. After the update, $\mathcal{D} = \{\mathcal{A}_1, \mathcal{A}_2, \mathcal{A}_3, \mathcal{A}_4, \mathcal{A}_5, \mathcal{A}_6\}$ and $\mathcal{V}_+ = \{\mathcal{A}_5\}$. $\mathcal{V}_-$ is updated by setting $\mathcal{V}_- = \{\mathcal{A}_5\}$.

(9). Set $\mathcal{B}_- = \mathcal{A}_5$ and $\mathcal{S}_- = \mathcal{S}_5$. For all $\boldsymbol{s} \in \mathcal{S}_-$, the single basis $\{\xi_{665}\}$ is selected. Thus, $\mathcal{A}_5$ is updated by setting $\mathcal{A}_5 = \mathcal{B}_- \cup \{\xi_{665}\} = \{\xi_1, \xi_2, \xi_{863}, \xi_{379}, \xi_{709}, \xi_{607}, \xi_{835}, \xi_{875}, \xi_{932}, \xi_{984}, \xi_{970}, \xi_{665}\}$. $\mathcal{S}_5$ stays the same. After the update, $\mathcal{D} = \{\mathcal{A}_1, \mathcal{A}_2, \mathcal{A}_3, \mathcal{A}_4, \mathcal{A}_5, \mathcal{A}_6\}$, $\mathcal{V}_+ = \{\mathcal{A}_5\}$, and $\mathcal{V}_- = \{\mathcal{A}_5\}$.

(10). Set $\mathcal{B}_- = \mathcal{A}_5$ and $\mathcal{S}_- = \mathcal{S}_5$. For all $\boldsymbol{s} \in \mathcal{S}_-$, none of the BICs of the candidate models is lower than $\mathrm{BIC}_{\boldsymbol{s}}(\mathcal{B}_-)$, so $\mathcal{A}_5$ completes forward selection, and is removed from $\mathcal{V}_+$. After the update, $\mathcal{D} = \{\mathcal{A}_1, \mathcal{A}_2, \mathcal{A}_3, \mathcal{A}_4, \mathcal{A}_5, \mathcal{A}_6\}$ and $\mathcal{V}_+ = \emptyset$. $\mathcal{V}_-$ is updated by setting $\mathcal{V}_- = \emptyset$. Therefore, the whole forward selection process has ended.

We can see from above that whenever new sets are added to $\mathcal{D}$ and $\mathcal{V}$, the IDF sets for these new models together with those for the updated current model form a partition of the IDF set for $\mathcal{B}_-$. For example, at iteration (2), $\mathcal{S}_{1,0}$ and $\mathcal{S}_{1,1}$ form a partition of $\mathcal{S}_-$. The IDF sets for all the sets in $\mathcal{D}$ at iterations (1), (2), (3), and (5) are shown in Figure 3.2. We can clearly see that as the number of models in $\mathcal{D}$ increases, $\mathcal{S}$ is divided into more and more parts, where each part corresponds to a unique model in $\mathcal{D}$. Also, the boundaries between different parts in $\mathcal{S}$ need not be parallel or perpendicular, as can be seen from iteration (3).

We can also use the tree diagram to summarize the forward selection process described above, as shown in Figure 3.1. Here the edges are labeled with the iteration numbers, and each node represents a unique model with its IDF set. The set $\mathcal{V}_-$ at each round corresponds to the collection of internal nodes with the same depth in the tree. For example, after iteration (3), $\mathcal{V}_-$ becomes $\{\mathcal{A}_2, \mathcal{A}_3, \mathcal{A}_4\}$, which can be represented by the collection of all three nodes with depth equal to 3 in the tree diagram. After each model in $\mathcal{V}_-$ is updated at iteration (4) through (6), $\mathcal{V}_-$ is updated by setting $\mathcal{V}_- = \{\mathcal{A}_5, \mathcal{A}_6\}$, since the nodes containing $\mathcal{A}_5$ and $\mathcal{A}_6$ are the only two internal nodes at the next level of the tree.

At each iteration, new basis functions that are added to the current model are colored in cyan. For example, from the diagram we can see that at iteration (1), the pair of bases $\{\xi_{863}, \xi_{379}\}$ is selected and added to $\mathcal{A}_1$. Each end node is red-filled and corresponds to the end of forward selection for each model. Thus, it can be seen that $\mathcal{A}_1$ through $\mathcal{A}_6$ completes the forward selection at iteration $(2), (4), (5), (6), (10)$, and (8) respectively. In addition, when a node has multiple branches, this means the IDF set of the current model is partitioned into multiple parts, where each part leads to a unique model (the child node) after the iteration.

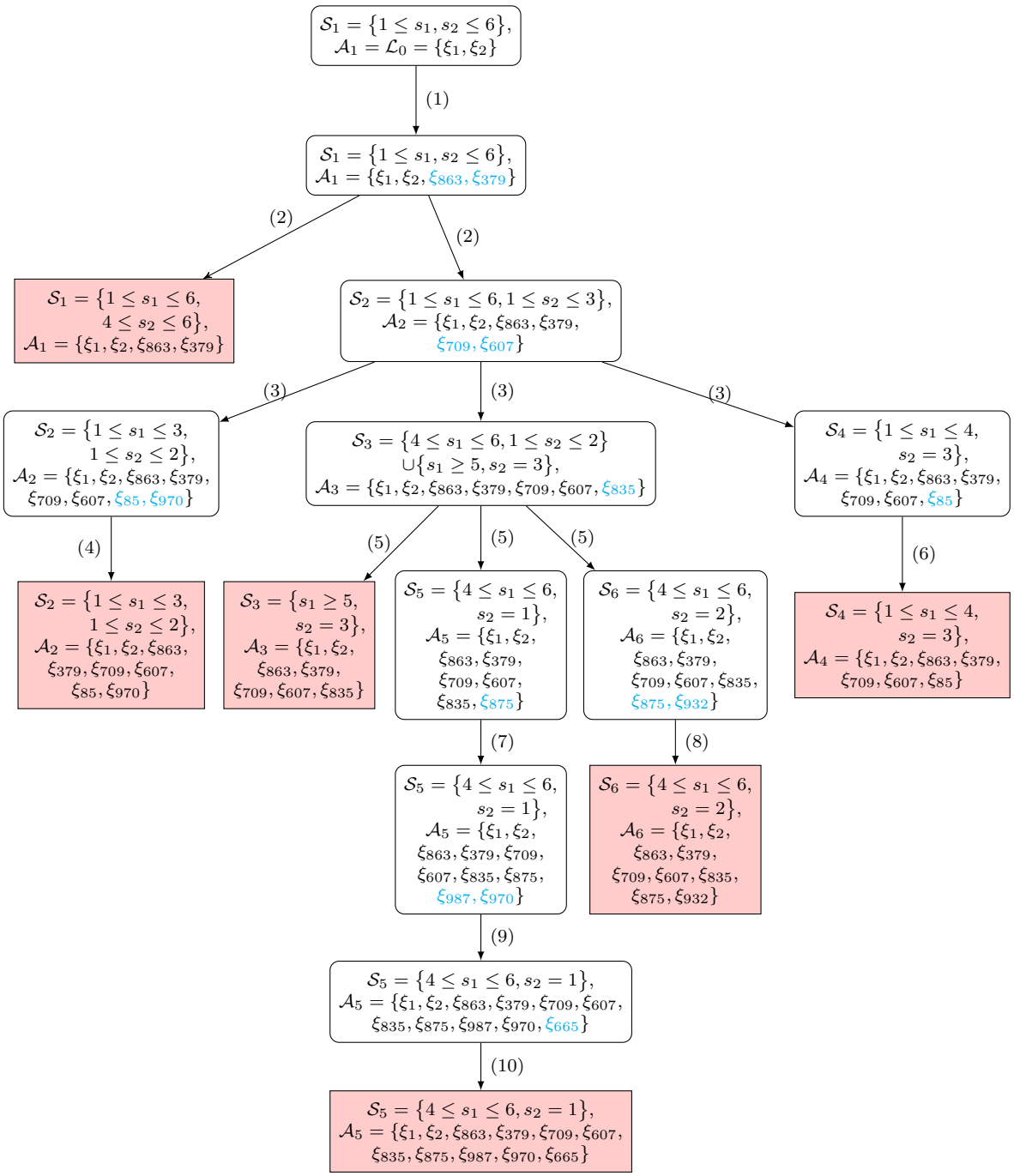
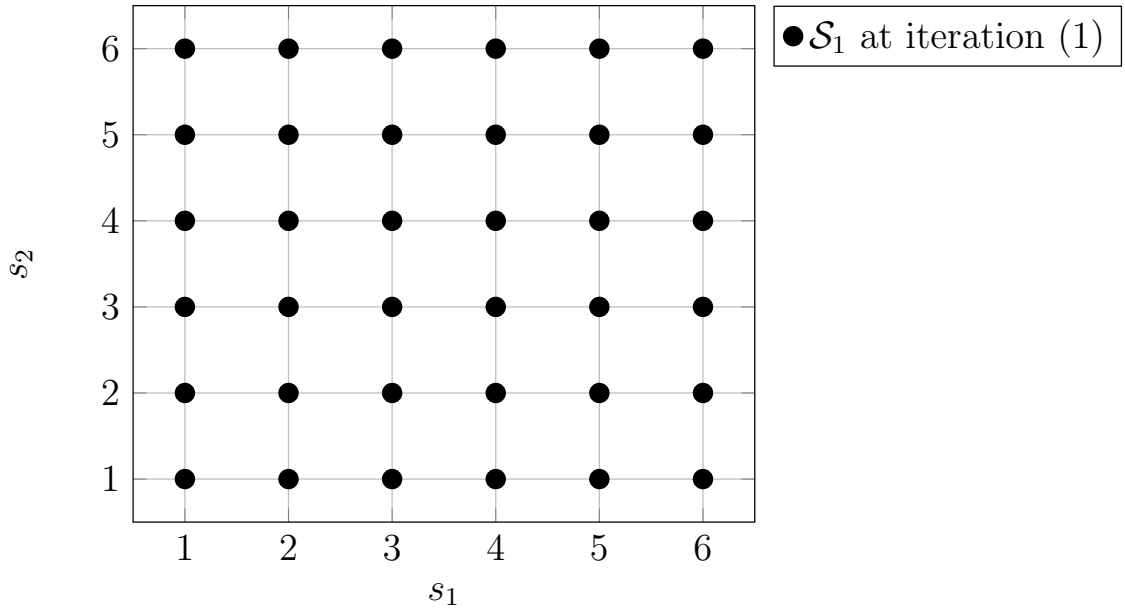
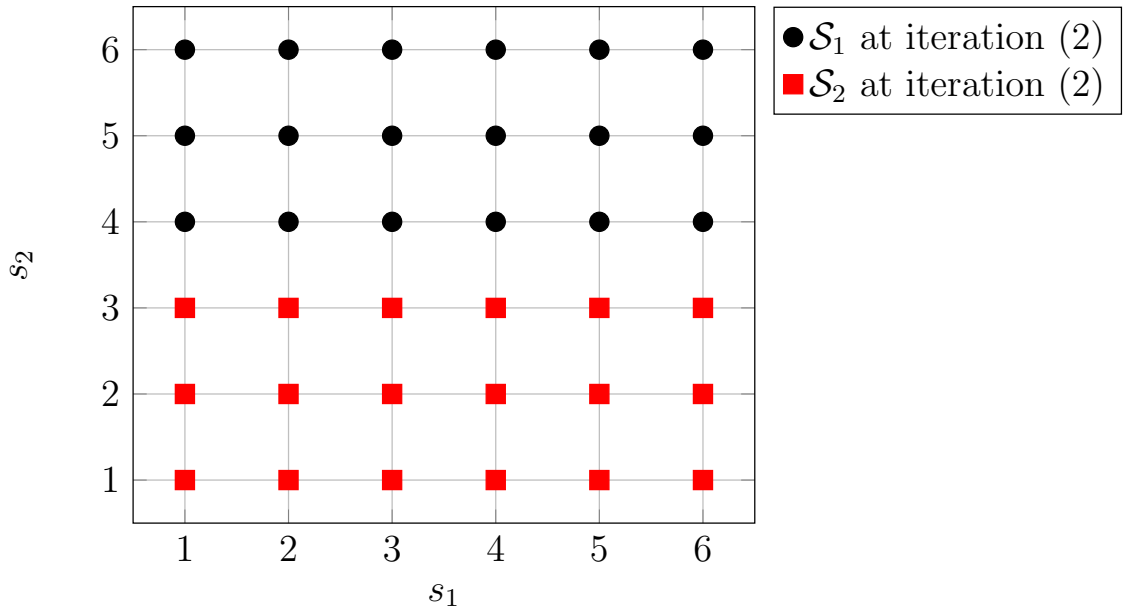


Figure 3.1: Tree diagram of the forward selection process for a data set from the Blocks-Curves example. Note that s_1 and s_2 take integer values from 1 to 6 in this illustration. The numbers in parentheses on left or right side of arrow represent iteration number in the searching procedure. Each end node is red-filled. New basis functions that are selected at each iteration are colored in cyan.


$\mathcal{S}_1$ at iteration (1)
s_2
s_1
1
2
3
4
5
6


$\mathcal{S}_1$ at iteration (2)
$\mathcal{S}_2$ at iteration (2)
s_2
s_1
1
2
3
4
5
6

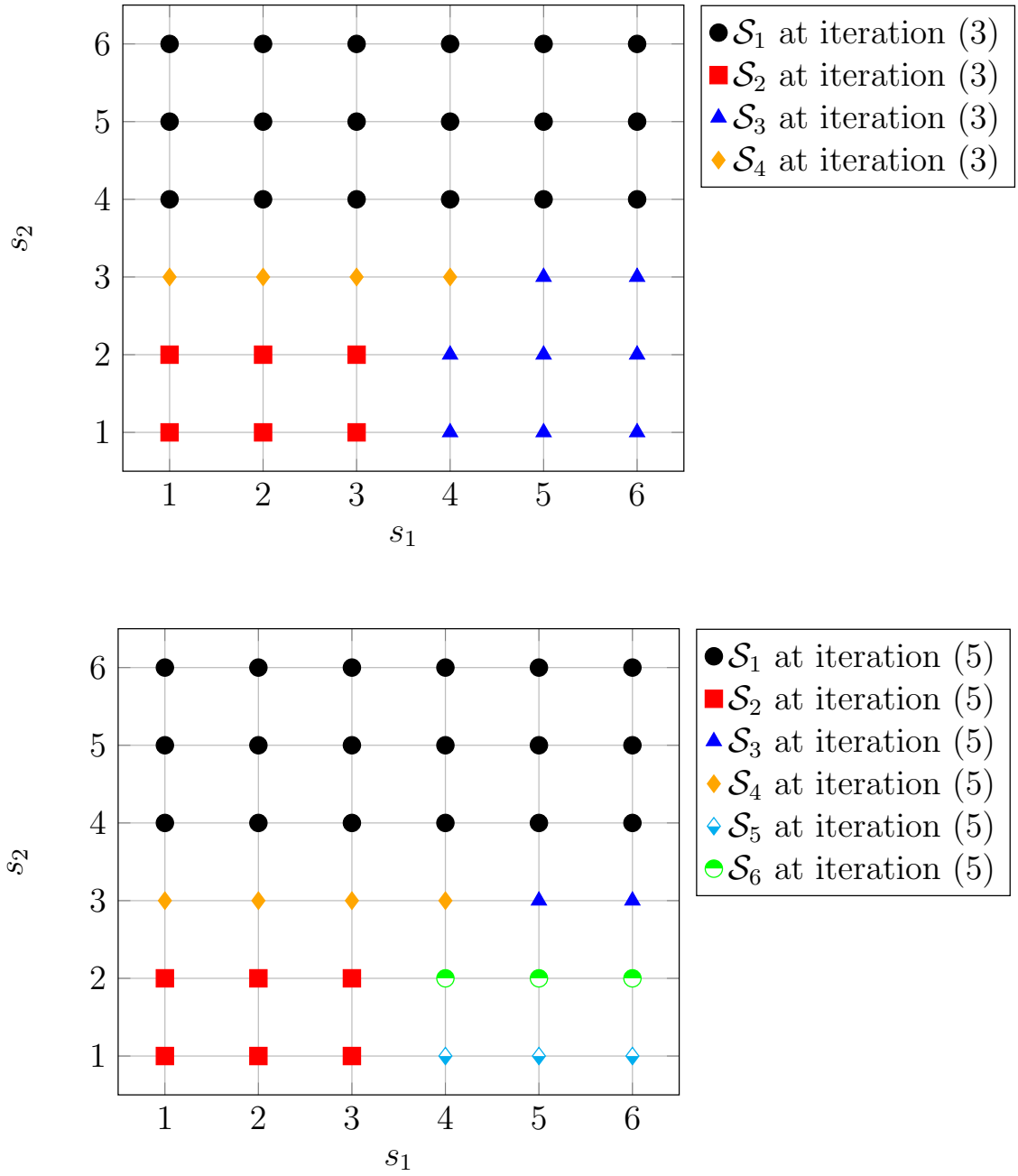


Figure 3.2: The sets of IDF tuning vectors for all the models in $\mathcal{D}$ at different iterations for a data set from the Blocks-Curves example

3.5 Selection of IDFs

After the forward selection process is completed, the look-ahead procedure returns a set of models $\mathcal{D}$. Each model has its own IDF set which contains at least one IDF tuning vector in the IDF space $\mathcal{S}$. Clearly if $|\mathcal{D}| < M_d$, then $\{\mathcal{S}_1, \ldots, \mathcal{S}_{|\mathcal{D}|}\}$ forms a partition of $\mathcal{S}$. Otherwise $\{\mathcal{S}_1, \ldots, \mathcal{S}_{|\mathcal{D}|}\}$ are disjoint subsets of $\mathcal{S}$. Now the question is how to select one of the models to be our final model.

Selecting a model in $\mathcal{D}$ is equivalent to selecting one of the IDF tuning vectors in $\mathcal{S}$, because for a fixed IDF vector, the model is uniquely determined by the forward selection process. The role of the IDF tuning vector is similar to the role of the smoothing parameter λ in the smoothing spline model. By penalizing libraries differently via IDFs, our procedure controls the selection of each type of basis functions. For instance, for the Blocks-Curves example presented in Section 3.4, when the procedure correctly selects the suitable penalties on $\mathcal{L}_1$ and $\mathcal{L}_2$ with $s_1 \leq 4$ and $s_2 = 3$, model $\mathcal{A}_4 = \{\xi_1, \xi_2, \xi_{863}, \xi_{379}, \xi_{709}, \xi_{607}, \xi_{85}\}$ will be the final model and exactly three step functions with jump points close to 0.2, 0.4, and 0.7 are selected.

We will use k-fold cross validation to select the best IDF tuning vector in $\mathcal{S}$. The whole data set is split into k roughly equal-sized parts. For $v = 1, ..., k$, we use the v-th part as the validation set and the other parts combined as the training set. For a tuning vector $\boldsymbol{s} \in \mathcal{S}$, define its average cross validation error as:

$$\mathrm{CV}_{\boldsymbol{s}} = \frac{1}{k} \sum_{v=1}^{k} \left\{ \frac{1}{n_v} \sum_{i=1}^{n_v} \left(y_i^{[v]} - \hat{f}_{\boldsymbol{s}}^{[-v]}\left(\boldsymbol{x}_i^{[v]}\right) \right)^2 \right\}, \tag{3.69}$$

where $\left\{ \left(\boldsymbol{x}_i^{[v]}, y_i^{[v]}\right), 1 \leq i \leq n_v \right\}$ is the v-th validation set, and $\hat{f}_{\boldsymbol{s}}^{[-v]}$ is the ordinary least squares fit based on all observations except those in the v-th validation set, using the basis functions obtained from the forward selection for the fixed IDF tuning vector $\boldsymbol{s}$.

The best IDF tuning vector is chosen to minimize $\mathrm{CV}_{\boldsymbol{s}}$:

$$\boldsymbol{s}^* = \underset{\boldsymbol{s}\in\mathcal{S}}{\operatorname{argmin}}\, \mathrm{CV}_{\boldsymbol{s}}. \tag{3.70}$$

Let $\mathcal{S}^*$ be the IDF set in $\{\mathcal{S}_1, \ldots, \mathcal{S}_{|\mathcal{D}|}\}$ that contains $\boldsymbol{s}^*$, then the final model $\mathcal{A}^*$ is the model in $\mathcal{D}$ whose IDF set is $\mathcal{S}^*$.

For the same Blocks-Curves example presented in Section 3.4, here is the levelplot of the average cross validation error for each IDF tuning vector in $\big\{(s_1, s_2), s_1, s_2 \in \{1, 2, 3, 4, 5, 6\}\big\}$. We can see that given $s_1 \in \{1, 2, 3\}$ and $s_2 = 3$, the average cross validation error is the smallest. Thus, model $\mathcal{A}_4 = \{\xi_1, \xi_2, \xi_{863}, \xi_{379}, \xi_{709}, \xi_{607}, \xi_{85}\}$ is chosen to be the final model for this specific example.

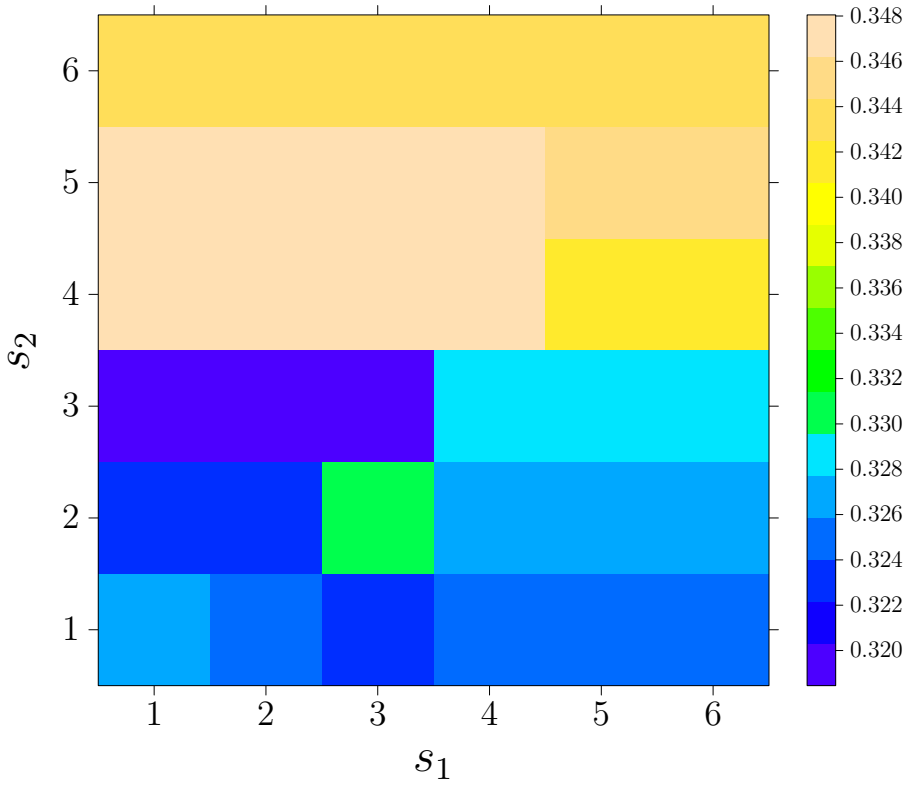


Figure 3.3: The average cross validation error for each IDF tuning vector, using a data set from the Blocks-Curves example.

For the choice of IDF tuning values for each library, one can first try values within a wide range, for example the $\{1, 2, 3, 4, 5, 6\}$ used above. Based on the IDF values selected by LAP, we can then fine-tune the IDFs and apply the LAP again. In the example above, LAP selects $s_1^* \in \{1, 2, 3\}$ and $s_2^* = 3$. For s_1, we can then specify a finer grid points between 1 and 3, and apply the LAP again to see if the basis functions selected will be different. For this example, the results stay the same.

3.6 The Whole Look-Ahead Procedure

The whole look-ahead procedure is summarized below.

The Look-Ahead Procedure

1. *Initialization:* Set $\mathcal{A}_1 = \mathcal{L}_0$, let M be an upper bound on the number of basis functions to be selected (including those in $\mathcal{L}_0$), and M_d be the maximum number of models to be generated in the forward selection. Let $\mathcal{S}$ be the IDF space. Set $\mathcal{D} = \mathcal{V}_+ = \mathcal{V}_- = \{\mathcal{A}_1\}$. Set $\mathcal{S}_1 = \mathcal{S}$, $\mathcal{B}_- = \mathcal{A}_1$ and $\mathcal{S}_- = \mathcal{S}_1$.

2. *Forward selection:*

   while $\mathcal{V}_+ \neq \emptyset$ and $\mathcal{V}_- \neq \emptyset$ **do**

   Find $\mathcal{A}_h \in \mathcal{V}_-$, which the first set in $\mathcal{V}_-$ that has not been updated.

   Set $\mathcal{B}_- = \mathcal{A}_h$, and set the IDF set of $\mathcal{A}_h$ as $\mathcal{S}_-$.

   Set $k = |\mathcal{B}_-| + 1$ and $d = |\mathcal{D}|$. Set $\Psi^- = \emptyset$.

   for $l = 1, \ldots, L$ **do**

   if $k = M$ **then**

   Select ψ_l^- using (3.1). Update Ψ^- with $\Psi^- \cup \{\psi_l^-\}$.

   else

Select ψ_l^- using (3.1). Update Ψ^- with $\Psi^- \cup \{\psi_l^-\}$.

Select $\psi_{\cdot|l}^-$ using (3.2). Update Ψ^- with $\Psi^- \cup \{\psi_l^-, \psi_{\cdot|l}^-\}$.

end if

end for

Let Ψ_j^- be the jth element of Ψ^-. Let $\mathcal{B}_j^- = \mathcal{B}_- \cup \Psi_j^-$ for $j = 1, \ldots, |\Psi^-|$.

Set $\mathcal{S}_{h,0} = \emptyset$, $\Psi_*^- = \emptyset$.

for $\boldsymbol{s} \in \mathcal{S}_-$ **do**

if $\mathrm{BIC}_{\boldsymbol{s}}(\mathcal{B}_-) \leq \min_{1\leq j\leq|\Psi^-|} \mathrm{BIC}_{\boldsymbol{s}}(\mathcal{B}_j^-)$ **then**

Set $\mathcal{S}_{h,0} = \mathcal{S}_{h,0} \cup \{\boldsymbol{s}\}$.

else

Solve $j^*(\boldsymbol{s}) = \underset{1\leq j\leq|\Psi^-|}{\operatorname{argmin}} \mathrm{BIC}_{\boldsymbol{s}}(\mathcal{B}_j^-)$, with $\mathrm{BIC}_{\boldsymbol{s}}$ defined in (3.6).
Set $\Psi_*^- = \Psi_*^- \cup \{\Psi_{j^*(\boldsymbol{s})}^-\}$.

end if

end for

if $\mathcal{S}_{h,0} = \mathcal{S}_-$ **then**

$\mathcal{A}_h$ is removed from $\mathcal{V}_+$. The IDF set of $\mathcal{A}_h$ is $\mathcal{S}_h = \mathcal{S}_{h,0}$.

else

Denote the elements in Ψ_*^- as $\Psi_{j_1^*}^-, \ldots, \Psi_{j_q^*}^-$. Generate the new models $\mathcal{A}_{h,1}, \ldots, \mathcal{A}_{h,q}$ using (3.67), and their IDF sets $\mathcal{S}_{h,1} \ldots \mathcal{S}_{h,q}$, where $\mathcal{S}_{h,i} = \left\{\boldsymbol{s} : \boldsymbol{s} \in \mathcal{S}_- \setminus \mathcal{S}_{h,0}, \mathrm{BIC}_{\boldsymbol{s}}(\mathcal{B}_{j_i^*}^-) = \min_{1\leq j\leq|\Psi^-|} \mathrm{BIC}_{\boldsymbol{s}}(\mathcal{B}_j^-)\right\}$, for $i = 1, \ldots, q$.

if $\mathcal{S}_{h,0} \subsetneq \mathcal{S}_-$ and $\mathcal{S}_{h,0} \neq \emptyset$ **then**

Remove $\mathcal{A}_h$ from $\mathcal{V}_+$.

Generate $\mathcal{A}_{d+i} = \mathcal{A}_{h,i}$ and $\mathcal{S}_{d+i} = \mathcal{S}_{h,i}$ for $i = 1, \ldots, q$.

Add the first $\min(q, M_d - d)$ of $\mathcal{A}_{d+1}, \ldots, \mathcal{A}_{d+q}$ to $\mathcal{D}$ and $\mathcal{V}_+$.

For $\mathcal{A}_j \in \mathcal{V}_+$, remove it from $\mathcal{V}_+$ if $|\mathcal{A}_j| = M$.

else

Update $\mathcal{A}_h$ by setting $\mathcal{A}_h = \mathcal{A}_{h,1}$. Update $\mathcal{S}_h$ by setting $\mathcal{S}_h = \mathcal{S}_{h,1}$.

if $q > 1$ **then**

Generate $\mathcal{A}_{d+i-1} = \mathcal{A}_{h,i}$ and $\mathcal{S}_{d+i-1} = \mathcal{S}_{h,i}$ for $i = 2, \ldots, q$.

Add the first $\min(q-1, M_d - d)$ of $\mathcal{A}_{d+1}, \ldots, \mathcal{A}_{d+q-1}$ to $\mathcal{D}$ and $\mathcal{V}_+$. For $\mathcal{A}_j \in \mathcal{V}_+$, remove it from $\mathcal{V}_+$ if $|\mathcal{A}_j| = M$.

end if

end if

end if

if all sets in $\mathcal{V}_-$ have been updated **then**

Update $\mathcal{V}_-$ by setting $\mathcal{V}_- = \mathcal{V}_+$.

end if

end while

3. *Elimination:* Compute the average cross validation error $\mathrm{CV}_{\boldsymbol{s}}$ in (3.69) for each $\boldsymbol{s} \in \mathcal{S}$. Find the $\boldsymbol{s}^*$ in (3.70). The final model is $\mathcal{A}_{h^*}$ such that $\boldsymbol{s}^* \in \mathcal{S}_{h^*}$.

3.7 Bootstrap Confidence Intervals

The nonparametric bootstrap confidence intervals have been well studied. The main difficulty in constructing them is due to the bias, which can not be estimated consistently using the bootstrap method. Consider the nonparametric regression model

$$y_i = f(\boldsymbol{x}_i) + \varepsilon_i, \quad \varepsilon_i \overset{iid}{\sim} N(0, \sigma^2), \quad i = 1, \ldots, n, \tag{3.71}$$

where $\boldsymbol{x}_i$'s are design points with arbitrary domain $\mathcal{X}$. As Cummins et al. (2001) has pointed out, by minimizing the average mean squared error (averaged across design points), the bias and variance of the estimate $\hat{f}$ are deliberately balanced, so $\text{Bias}(\hat{f}(\boldsymbol{x}_i))^2/\text{Var}(\hat{f}(\boldsymbol{x}_i))$ converges to a fixed ratio. When this ratio is large, the confidence interval will be consistently centered at the wrong place, causing low coverage. The bias of $\hat{f}$ tends to be big especially at points where f has sharp curvature or sudden jumps, and the coverage probability will be low at these points.

There are two common approaches to overcome this problem. One is by oversmoothing and inducing an explicit or implicit bias estimator, for example see Härdle and Bowman (1988), Härdle and Marron (1991), Hall (1992a), Eubank and Speckman (1993), Sun and Loader (1994), Härdle et al. (1995), and Xia (1998). The other approach is using undersmoothing to reduce the impact of bias, for example see Bjerve et al. (1985), Hall (1992b), Neumann (1995), Neumann and Polzehl (1998), Picard and Tribouley (2000), Claeskens and Keilegom (2003), McMurry and Politis (2008). See Hall and Horowitz (2013) for a more comprehensive review. In the smoothing splines literature, Wang and Wahba (1995) compared the average coverage probability of various bootstrap confidence intervals, such as the percentile interval (Efron (1982)) and the pivotal interval (Efron (1981)), with Bayesian confidence intervals (Wahba (1983)) for smoothing splines with

Gaussian data, and found they have similar performance. Cummins et al. (2001) constructed pointwise confidence intervals by selecting local smoothing parameters to yield more uniform pointwise coverage.

For the look-ahead procedure, we construct the bootstrap confidence intervals using the undersmoothing approach. First, the LAP is applied to the data $\{(\boldsymbol{x}_i, y_i), i = 1 \ldots, n\}$ to obtain the estimate $\hat{f}$. Recall that in LAP, criteria such as the BIC and AIC are used for selecting basis functions, and in order to avoid overfitting, the forward selection process for a candidate model stops once BIC or AIC starts to increase. Therefore, the estimate $\hat{f}$ tends to have big bias, especially at places where f has sharp curvature or sudden jumps. To reduce the bias, we use the undersmoothing approach to obtain a "big" model. Sklar et al. (2013) found that for the BSML procedures the cross validation method generally tends to lead to a model that overfits. We will use the same method for the look-ahead procedure.

Let $\boldsymbol{s}^*$ be the IDF tuning vector that minimizes the average cross validation error and leads to an estimate $\hat{f}$, as defined in (3.70). The same LAP forward selection routine in Section 3.2 is then applied again with the IDF vector fixed at $\boldsymbol{s}^*$. However, this time instead of stopping the forward selection once the BIC or AIC starts to increase, we let it continue until the number of basis functions in the model reaches M. The forward selection starts with $\mathcal{B}^{(1)} = \mathcal{L}_0$ at the first iteration. At the jth iteration, for $j = 2, 3, \ldots$, LAP sets $\mathcal{B}_- = \mathcal{B}^{(j-1)}$, then selects the set of basis functions $\Psi^-_{j^*}$ by following the same selection routine in (3.1), (3.2), (3.5), (3.6), (3.7), (3.9), (3.12), (3.13), and sets $\mathcal{B}^{(j)} = \mathcal{B}^{(j-1)} \cup \Psi^-_{j^*}$. This process continues until it reaches iteration D such that $|\mathcal{B}^{(D)}| = M$. Therefore, we have D candidate models: $\mathcal{B}^{(1)}, \ldots, \mathcal{B}^{(D)}$. Then the k-fold cross validation method is used to select one of them as the final model. The average

cross validation error for model $\mathcal{B}^{(j)}$, $j = 1, \ldots, D$, is defined as:

$$\mathrm{CV}\big(\mathcal{B}^{(j)}\big) = \frac{1}{k}\sum_{v=1}^{k}\left\{\frac{1}{n_v}\sum_{i=1}^{n_v}\Big(y_i^{[v]} - \hat{f}_{(j)}^{[-v]}\big(\boldsymbol{x}_i^{[v]}\big)\Big)^2\right\}, \tag{3.72}$$

where $\left\{\big(\boldsymbol{x}_i^{[v]}, y_i^{[v]}\big), 1 \le i \le n_v\right\}$ is the v-th validation set, and $\hat{f}_{(j)}^{[-v]}$ is the ordinary least squares fit based on all observations except those in the v-th validation set, using the basis functions contained in $\mathcal{B}^{(j)}$. The final model $\tilde{\mathcal{B}}$ is chosen to minimize the average k-fold cross validation error:

$$\tilde{\mathcal{B}} = \underset{\mathcal{B}^{(j)} \in \left\{\mathcal{B}^{(1)}, \ldots, \mathcal{B}^{(D)}\right\}}{\operatorname{argmin}} \mathrm{CV}\big(\mathcal{B}^{(j)}\big). \tag{3.73}$$

Once $\tilde{\mathcal{B}}$ has been chosen, denote $\tilde{f}$ as the ordinary least squares fit using the basis functions contained in $\tilde{\mathcal{B}}$. Then B bootstrap samples $\tilde{\boldsymbol{y}}_b = (\tilde{y}_{1,b}, \ldots, \tilde{y}_{n,b})'$ for $b = 1 \ldots, B$ are generated according to

$$\tilde{y}_{i,b} = \tilde{f}(\boldsymbol{x}_i) + \tilde{\varepsilon}_{i,b}, \quad \tilde{\varepsilon}_{i,b} \overset{iid}{\sim} N(0, \hat{\sigma}^2), \quad i = 1, \ldots, n;\ b = 1, \ldots, B, \tag{3.74}$$

where $\hat{\sigma}^2$ is an estimate of σ^2. Typically a difference-based estimator of σ^2 is used, for example see Rice (1984), Gasser et al. (1986), and Tong and Wang (2005). For the look-ahead procedure, the Rice estimator defined in (2.13) is used as the default. Alternatively, the bootstrap samples can be generated by resampling the residuals with replacement. In the look-ahead procedure, only the parametric bootstrap given in (3.74) is implemented.

For each bootstrap sample $\tilde{\boldsymbol{y}}_b$, $b = 1, \ldots, B$, the same forward selection routine mentioned above is applied to obtain the sequence of candidate models $\mathcal{B}_b^{(1)}, \ldots, \mathcal{B}_b^{(D_b)}$. To select one of them as the final model, we use the average square error (ASE) instead of the average cross validation error as the model elimination criterion, because the true

function $\tilde{f}$ in the generation of the bootstrap samples is known. The ASE of the model $\mathcal{B}_b^{(j)}$ is defined as:

$$\mathrm{ASE}\big(\mathcal{B}_b^{(j)}\big) = \frac{1}{n}\sum_{i=1}^{n}\Big(\hat{f}_b^{(j)}(\boldsymbol{x}_i) - \tilde{f}(\boldsymbol{x}_i)\Big)^2, \tag{3.75}$$

for $j = 1, \ldots, D_b$, where $\hat{f}_b^{(j)}$ is the ordinary least squares fit using the basis functions contained in $\mathcal{B}_b^{(j)}$. For the bth bootstrap sample, the final model $\mathcal{B}_b^*$ is selected to minimize the ASE, i.e.,

$$\mathcal{B}_b^* = \underset{\mathcal{B}_b^{(j)} \in \{\mathcal{B}_b^{(1)}, \ldots, \mathcal{B}_b^{(D)}\}}{\operatorname{argmin}} \mathrm{ASE}\big(\mathcal{B}_b^{(j)}\big). \tag{3.76}$$

Denote $\hat{f}_b^*$ as the ordinary least squares fit using the basis functions contained in $\mathcal{B}_b^*$, for $b = 1 \ldots, B$. Then the $(1-\alpha)\%$ confidence interval of $f(\boldsymbol{x})$ is $\big(\hat{f}_{\alpha/2}, \hat{f}_{(1-\alpha/2)}\big)$, where $\hat{f}_{\alpha/2}$ and $\hat{f}_{(1-\alpha/2)}$ are the $\alpha/2$ and $(1-\alpha/2)$ percentiles of $\big(\hat{f}_1^*(\boldsymbol{x}), \ldots, \hat{f}_B^*(\boldsymbol{x})\big)$.

The simulation results for the bootstrap confidence intervals are presented in Section 4.2.

3.8 R Functions for the Look-Ahead Procedure

The look-ahead procedure has been implemented in `R`. Here we briefly introduce the input arguments and output values of the `R` functions that can be used for applying LAP and performing predictions.

3.8.1 LAP.R

Description

This function implements the look-ahead procedure (LAP). The program adaptively select one or a pair of basis functions from different libraries according to certain criterion such as the BIC. Each library is associated with an IDF, which is treated as a tuning parameter. K-fold cross validation is used for selecting the best IDF values and the final model.

Usage

```
LAP(y, baseslist, maxbas=30, sub.maxbas=c(), idfs, maxmod=nrow(idfs),
    nfolds=10, criterion="BIC")
```

Arguments

- `y`: response vector.
- `baseslist`: list of libraries of basis functions, where each element corresponds on the list corresponds to a library. The first library is the null library $\mathcal{L}_0$, the second library is $\mathcal{L}_1$ and so on. Each library should be a matrix, where each column corresponds to a basis function evaluated at the design points.
- `maxbas`: maximum number of bases allowed to be selected. This is the M in the LAP algorithm.

- `sub.maxbas`: a vector with length equal to the number of nonnull libraries. Each element specifies the maximum number of bases to be preselected for each nonnull library.

- `idfs`: matrix of IDF tuning values with $L+1$ columns, where each row represents an IDF tuning vector. The IDF for $\mathcal{L}_0$ (the first column) should be fixed at 1.

- `maxmod`: maximum number of models allowed to be generated during the forward selection process. This is denoted as M_d in the LAP algorithm. The default is the total number of IDF tuning vectors, or equivalently, the number of rows in the matrix `idfs`.

- `nfolds`: the number of folds used in the cross validation.

- `criterion`: basis selection criterion. Available options are "AIC" and "BIC". Default is "BIC".

Value

- `bases.chosen`: a matrix with two columns. The second column contains indices of the basis functions selected by LAP. The first column contains the indices of the libraries to which basis functions in the second column belong, where "1" represents the null library, "2" represents $\mathcal{L}_1$, and so on.

- `cv.error`: the average cross validation error for each IDF tuning vector, see (3.72).

- `idfs.cv`: the IDF values that minimize average cross validation error, see (3.70).

- `fit`: the fitted values of the final model.

- `coef`: the coefficients in the final model.

- `sigma_hat`: estimated error standard deviation.

- `RSS`: residual sum of squares of the final model.
- `y`: response vector. Same as input.
- `baseslist`: list of libraries of basis functions. Same as input.
- `maxbas`: maximum number of bases allowed to be selected. Same as input.
- `idfs`: list of IDF tuning values for each library, including $\mathcal{L}_0$. Same as input.
- `maxmod`: maximum number of models allowed to be generated during the forward selection process. Same as input.
- `criterion`: basis selection criterion. Same as input.

3.8.2 predict.LAP.R

Description

This function is for making predictions using the output object from LAP.

Usage

```
predict.LAP (object, bases.include=NULL, new.baseslist=NULL,
             confint=F, alpha=0.05, sigma=NULL, bootrep=200)
```

Arguments

- `object`: output object from LAP.
- `bases.include`: a vector of length (number of null bases + number of nonnull libraries). The elements should be either 0 or 1, with 0 means the corresponding basis or library will not be included in the prediction, and 1 otherwise.
- `new.baseslist`: the list of libraries used in the prediction. If the original list in the LAP object is used, the prediction will be same as the fitted values in the LAP object.

- `confint`: True or False. If True, a confidence interval with significance level alpha will be computed. The default is False.
- `alpha`: the significance level for the confidence interval. Default is 0.05.
- `sigma`: the error standard deviation used in constructing the bootstrap sample. The Rice estimator is used as the default.
- `bootrep`: number of bootstrap samples used in constructing the confidence interval. Default is 200.

Value

- `fit`: predicted values.
- `fit.boot`: the fit of the model $\tilde{\mathcal{B}}$ obtained by using cross validation.
- `bases.boot`: the indices of the basis functions contained in the model $\tilde{\mathcal{B}}$.
- `bootfits`: a matrix which contains all the bootstrap fitted values used in constructing the confidence interval as its columns. NULL if `confint=F`.
- `boot.y`: a matrix which contains the bootstrap samples generated for the response variable as its columns. NULL if `confint=F`.
- `bootbases.all`: a list which contains the indices of all the basis functions selected for each bootstrap sample. NULL if `confint=F`.
- `bootbases.pred`: a list which contains the indices of selected basis functions, among all the wanted ones as specified in `bases.include`, for each bootstrap sample. NULL if `confint=F`.
- `lower`: the lower bound of the confidence interval. NULL if `confint=F`.

- `upper`: the upper bound of the confidence interval. NULL if `confint=F`.
- `sigma`: the error standard deviation used in constructing the bootstrap sample.

3.8.3 stdz.R

Description

This function is used to standardize the basis functions (columns) in a design matrix, so that each basis functions evaluated at the design points has norm one.

Usage

```
stdz(X)
```

Arguments

- `X`: matrix of basis functions. Each column of X is a basis function evaluated at the design points.

Value

- matrix with standardized basis functions as its columns.

3.8.4 An Example

We use one simulated data set from the Block-Curves example in Section 3.4 to illustrate the usage of these R functions. The R code is shown below, where the R file `LAP.R` contains `LAP` and `predict.LAP` functions. Here the `stdz` function is used to numerically stabilize the computation. The input matrix in the `stdz` function is standardized by column so that each column has norm one.

```
library(assist)
source("LAP.R")

set.seed(82)
```

```
n <- 512
x <- seq(from=1/n, to=1, by=1/n)
fx <- 1*(x-0.2 >= 0 & x-0.4 < 0) - x^10*(x-0.7 >= 0) +
      exp((x+0.5)^2)*(x-0.4 >= 0 & x-0.7 < 0)

## Libraries of basis functions:
loc <- x[-c(1,2,3,n-2,n-1,n)]
pt <- rep(1,n)%o%loc
L0 <- stdz(cbind(1, x))
L1 <- stdz((cubic(s = x, t = loc)))
L2 <- stdz((1*(x-pt)>0))
baseslist <- list(L0, L1, L2)

## Look-Ahead Procedure:
y <- fx + rnorm(n, mean=0, sd=sd(fx)/3)

nlib <- length(baseslist)
idfs <- append(list(1),rep(list(c(1:6)),nlib-1))
idfs <- as.matrix(expand.grid(idfs))
fit.lap <- LAP(y, baseslist, maxbas=30, idfs=idfs, criterion="BIC")


## 95% bootstrap confidence interval:
pred.lap <- predict.LAP(fit.lap, bases.include=c(1,1,1,1),
                        new.baseslist=baseslist, confint=T, bootrep=200)


## Plot of the LAP fit with its 95% confidence interval:
plot(x, y, col="gray")
lines(x, fx, type="l", lwd=2)
lines(x, fit.lap$fit, col=2, lty=1, lwd=2)
lines(x, pred.lap$lower, col=2, lty=2, lwd=2)
lines(x, pred.lap$upper, col=2, lty=2, lwd=2)
```

Figure 3.4 shows the LAP fit with its 95% bootstrap confidence interval for this example.

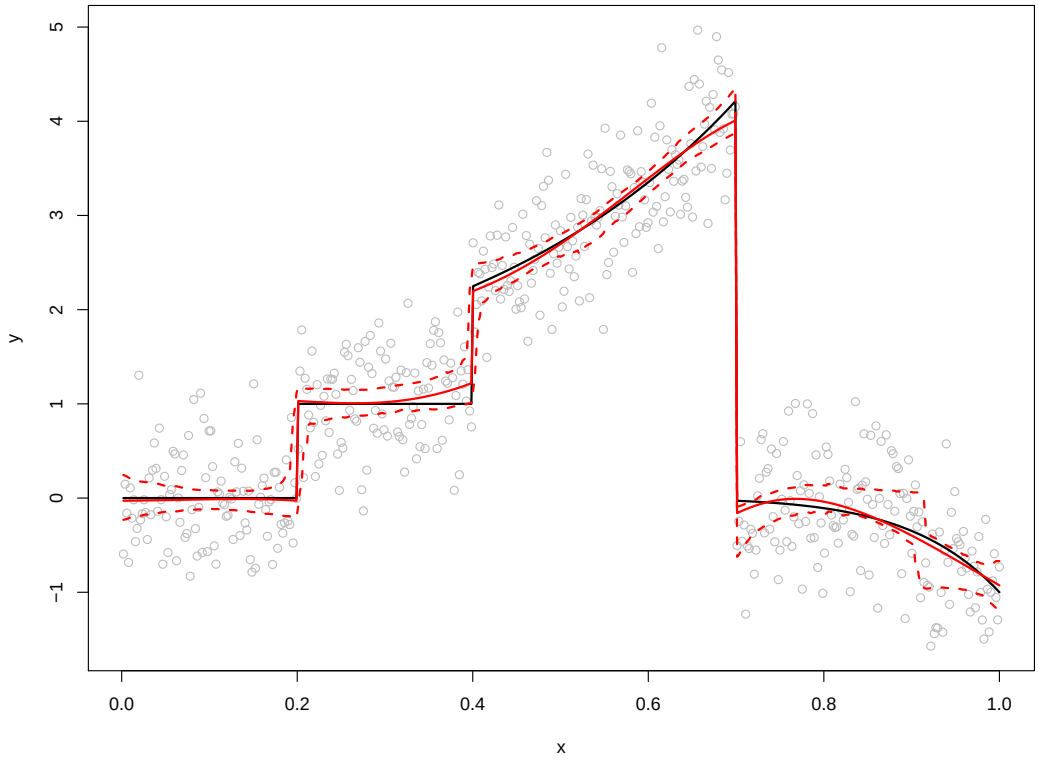


Figure 3.4: LAP fit of the Block-Curves example in Section 3.4. Gray dots are the simulated data points. Black solid line is true function. Red solid line is the LAP fit. Red dashed lines are the upper and lower bounds of the 95% bootstrap confidence interval based on 200 bootstrap samples.

Chapter 4

Simulations

4.1 Estimation of Univariate Functions

We performed simulations to compare the performances of the BSML procedures and our look-ahead procedure with the same six univariate examples presented in Section 2.2. See Table 2.1 for the true functions and the libraries of basis functions used for each example.

The design points used for all simulations are the grid points $\{x_i = i/n : i = 1, \ldots, n\}$, where the sample size $n \in \{256, 512, 1024\}$. The response variable is generated using (2.6), where σ is chosen such that SNR $\in \{1, 2, 3, 4, 5, 6\}$. The knots used in the libraries $\mathcal{P}_2, \mathcal{C}_2, \mathcal{T}_0, \mathcal{T}_2$ are grid points $\{j/n,\ j = 4, \ldots, n-3\}$. For all procedures, the maximal number of bases M is fixed at 30. For the look-ahead procedure, we used IDF spaces $S_i = \{1, 2, 3, 4, 5, 6\}$ for $i = 1, 2$. The maximal number of models can be returned after the forward selection, denoted by M_d, was set to be $|S_1| \times |S_2| = 36$, so that all new models generated during the forward selection are kept in $\mathcal{D}$. The 10-fold cross validation is used to select the final model in the look-ahead procedure. For each sample size, 100 random samples are simulated to evaluate the performances of the BSML procedures

and the look-ahead procedure. We use the MSE defined in (2.8) as the measure of performance. For convenience, we will use the acronym "LAP" when referring to the look-ahead procedure from now on. The simulation results are shown in Figure 4.1 – 4.9.

We can see that LAP's performance is comparable to that of BSML-C and BSML-S in terms of the MSE. For the Sine-Jumps and LW6 examples, on average LAP selected fewer basis function than BSML-C and BSML-S. Also, as we mentioned in Section 2.4, for the Heavisine example the basis functions from $\mathcal{L}_1$ and $\mathcal{L}_2$ need to be treated separately. Because of this, for the Heavisine example the performance of BSML-C is not as good as that of BSML-S and LAP across all SNR values and sample sizes. In terms of the computation speed, LAP has clear advantage over both BSML procedures. Table 4.1 – 4.3 lists the average CPU time of fitting each univariate example using BSML-C, BSML-S, and LAP for different samples sizes with SNR = 3. Without having to estimate the GDF at each forward selection step as BSML-C and BSML-S do, LAP manages to be about twice as fast as BSML-C , and more than 10 times faster than BSML-S.

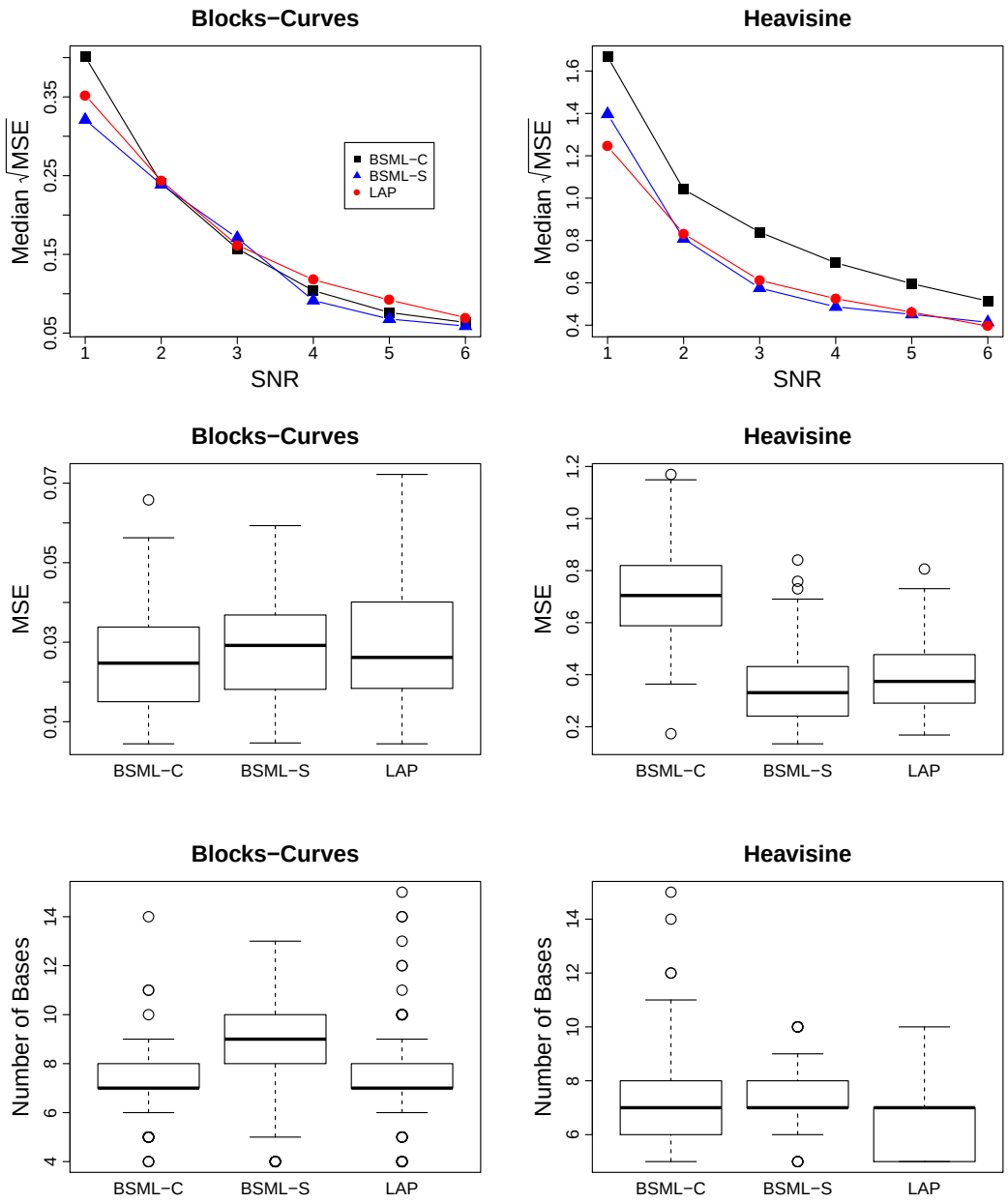


Figure 4.1: Simulation results for the Blocks-Curves and Heavisine examples with $n = 256$. The top panel shows the median $\sqrt{\text{MSE}}$ across different SNR values for both examples, where the symbols used are: ■ for BSML-C, ▲ for BSML-S, and ● for the look-ahead procedure. The middle panel shows the boxplots of the MSE when SNR $= 3$. The bottom panel shows the number of basis functions selected when SNR $= 3$.

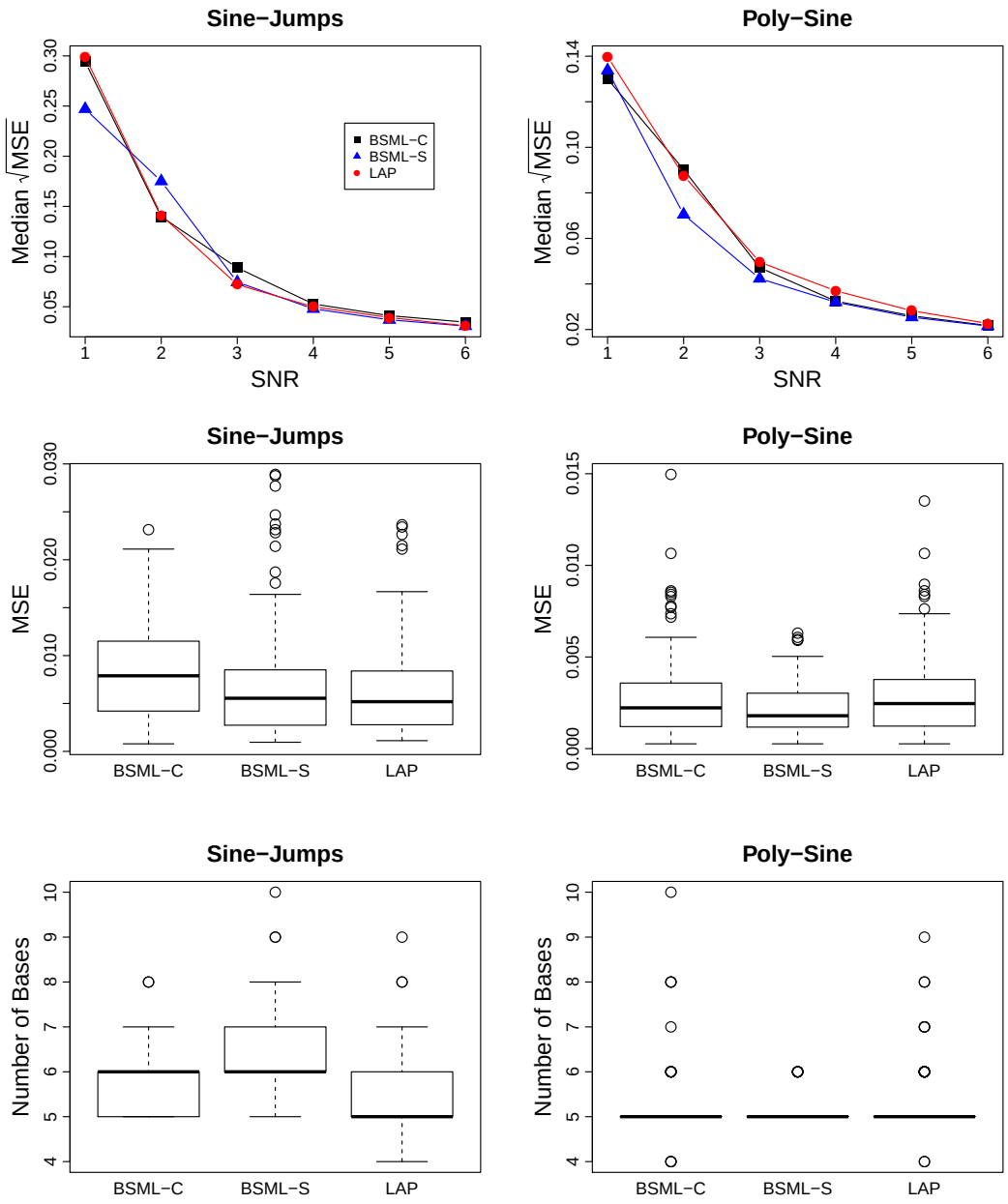


Figure 4.2: Simulation results for the Sine-Jumps and Poly-Sine examples with $n = 256$. The top panel shows the median $\sqrt{\text{MSE}}$ across different SNR values for both examples, where the symbols used are: ■ for BSML-C, ▲ for BSML-S, and ● for the look-ahead procedure. The middle panel shows the boxplots of the MSE when SNR $= 3$. The bottom panel shows the number of basis functions selected when SNR $= 3$.

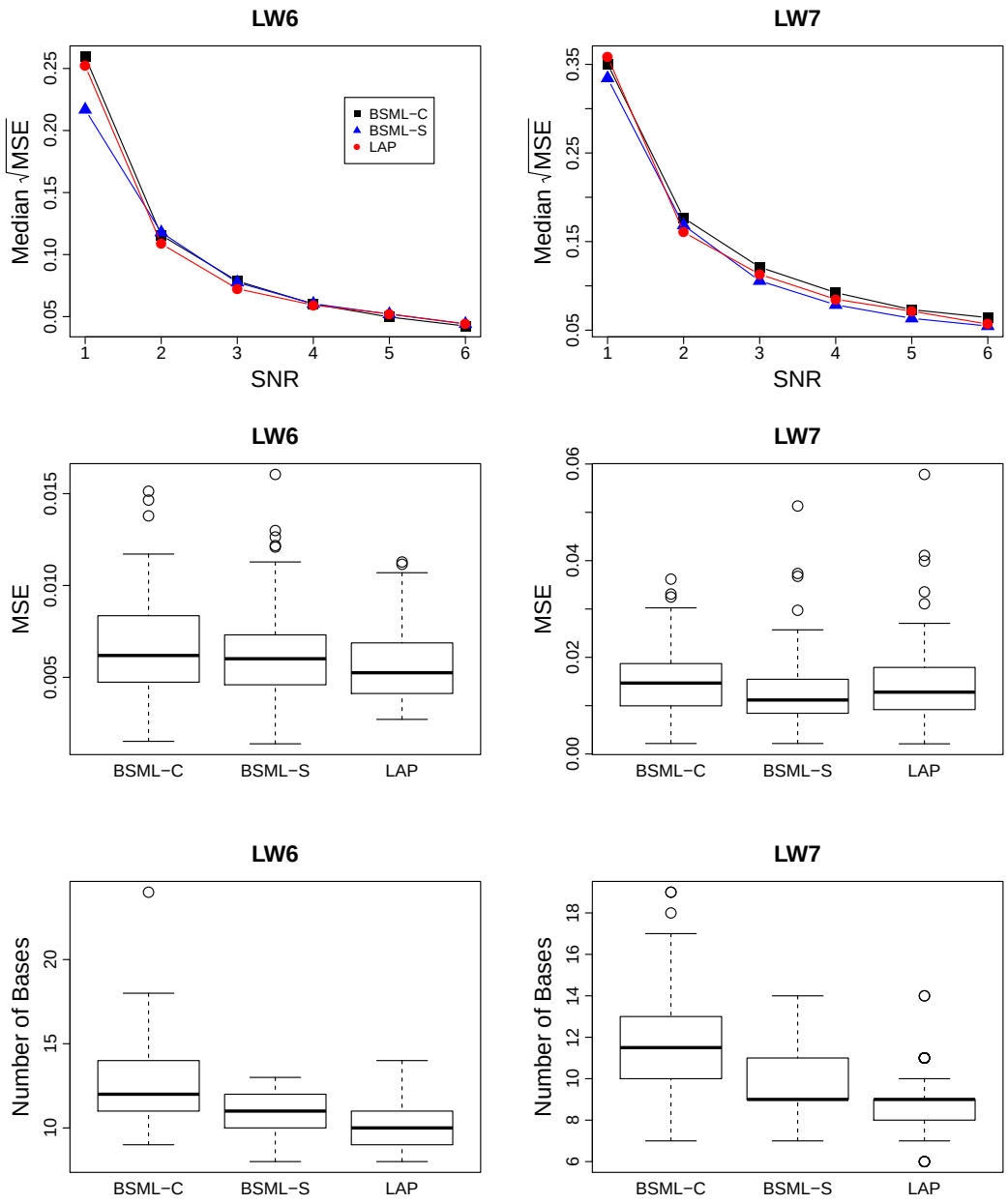


Figure 4.3: Simulation results for the LW6 and LW7 examples with $n = 256$. The top panel shows the median $\sqrt{\text{MSE}}$ across different SNR values for both examples, where the symbols used are: ■ for BSML-C, ▲ for BSML-S, and ● for the look-ahead procedure. The middle panel shows the boxplots of the MSE when SNR $= 3$. The bottom panel shows the number of basis functions selected when SNR $= 3$.

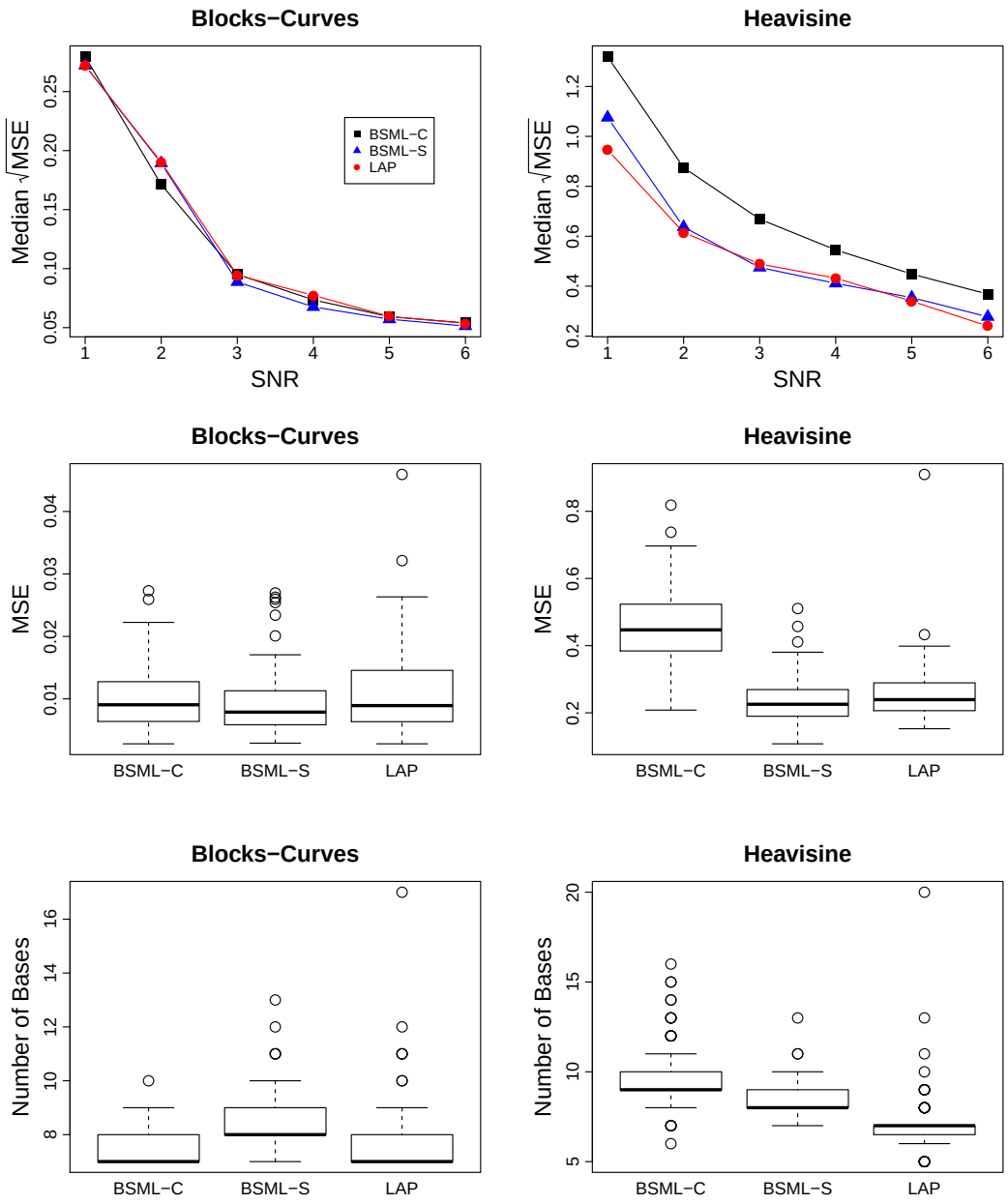


Figure 4.4: Simulation results for the Blocks-Curves and Heavisine examples with $n = 512$. The top panel shows the median $\sqrt{\text{MSE}}$ across different SNR values for both examples, where the symbols used are: ■ for BSML-C, ▲ for BSML-S, and ● for the look-ahead procedure. The middle panel shows the boxplots of the MSE when SNR $= 3$. The bottom panel shows the number of basis functions selected when SNR $= 3$.

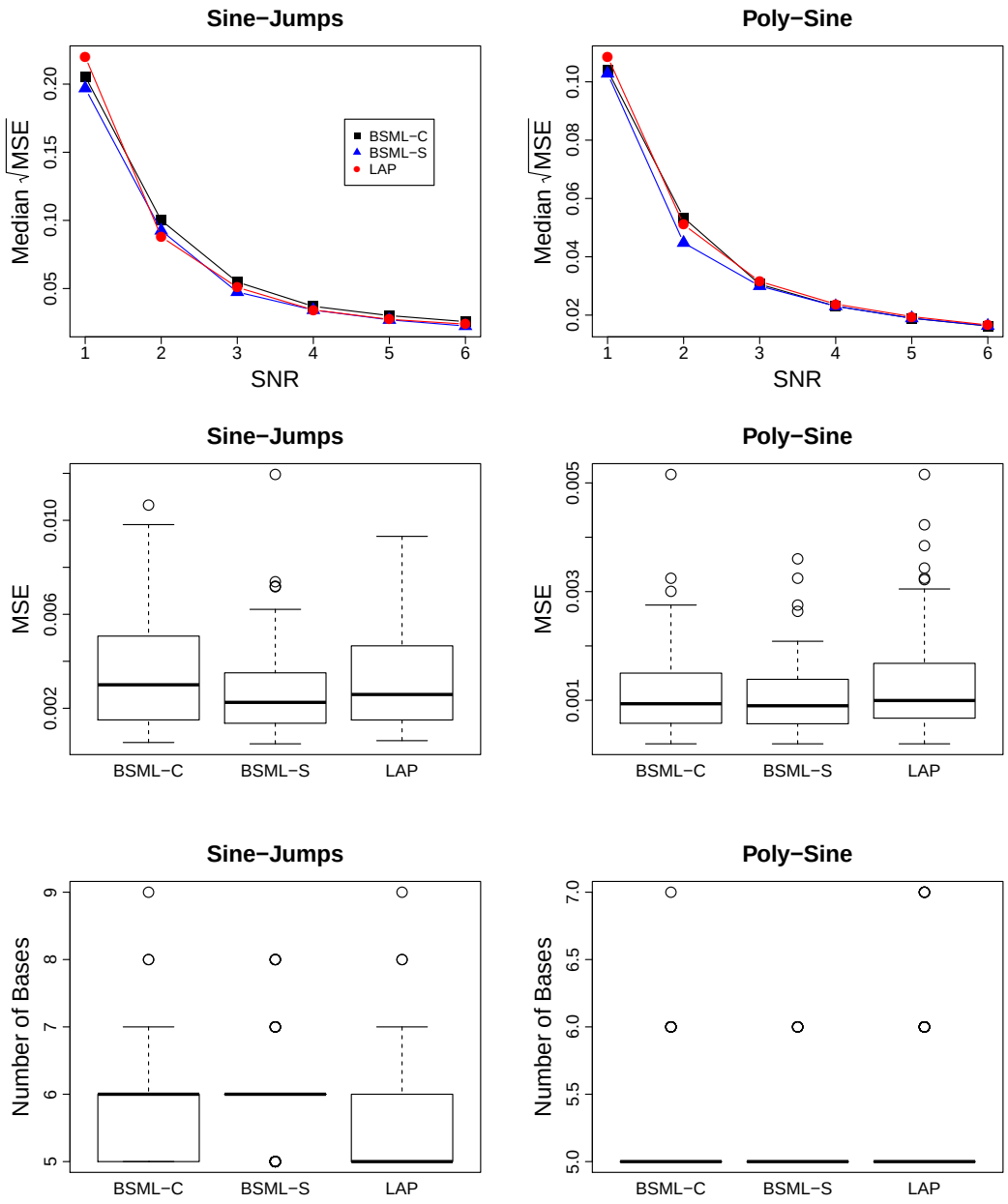


Figure 4.5: Simulation results for the Sine-Jumps and Poly-Sine examples with $n = 512$. The top panel shows the median $\sqrt{\text{MSE}}$ across different SNR values for both examples, where the symbols used are: ■ for BSML-C, ▲ for BSML-S, and ● for the look-ahead procedure. The middle panel shows the boxplots of the MSE when SNR $= 3$. The bottom panel shows the number of basis functions selected when SNR $= 3$.

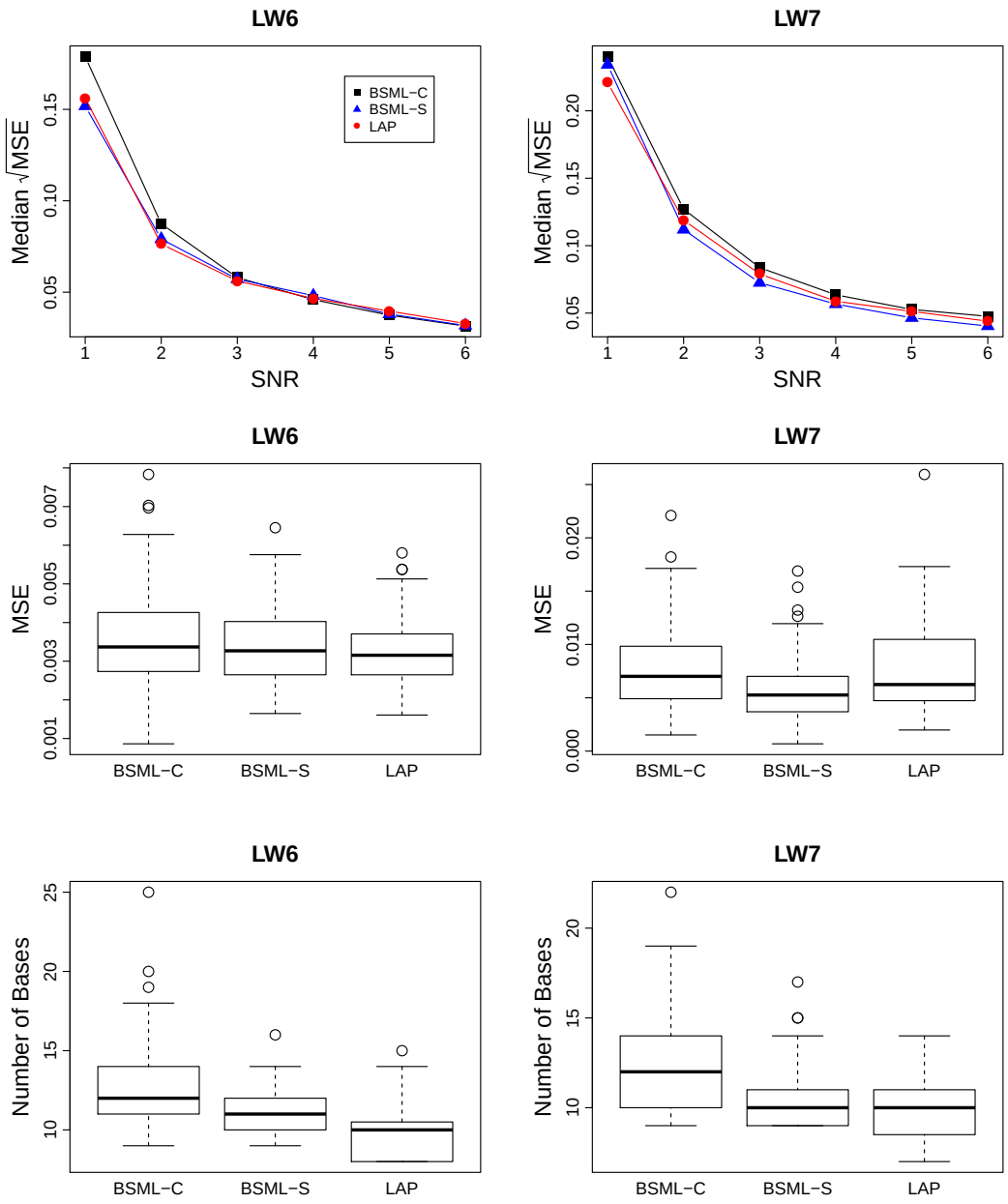


Figure 4.6: Simulation results for the LW6 and LW7 examples with $n = 512$. The top panel shows the median $\sqrt{\text{MSE}}$ across different SNR values for both examples, where the symbols used are: ■ for BSML-C, ▲ for BSML-S, and ● for the look-ahead procedure. The middle panel shows the boxplots of the MSE when SNR $= 3$. The bottom panel shows the number of basis functions selected when SNR $= 3$.

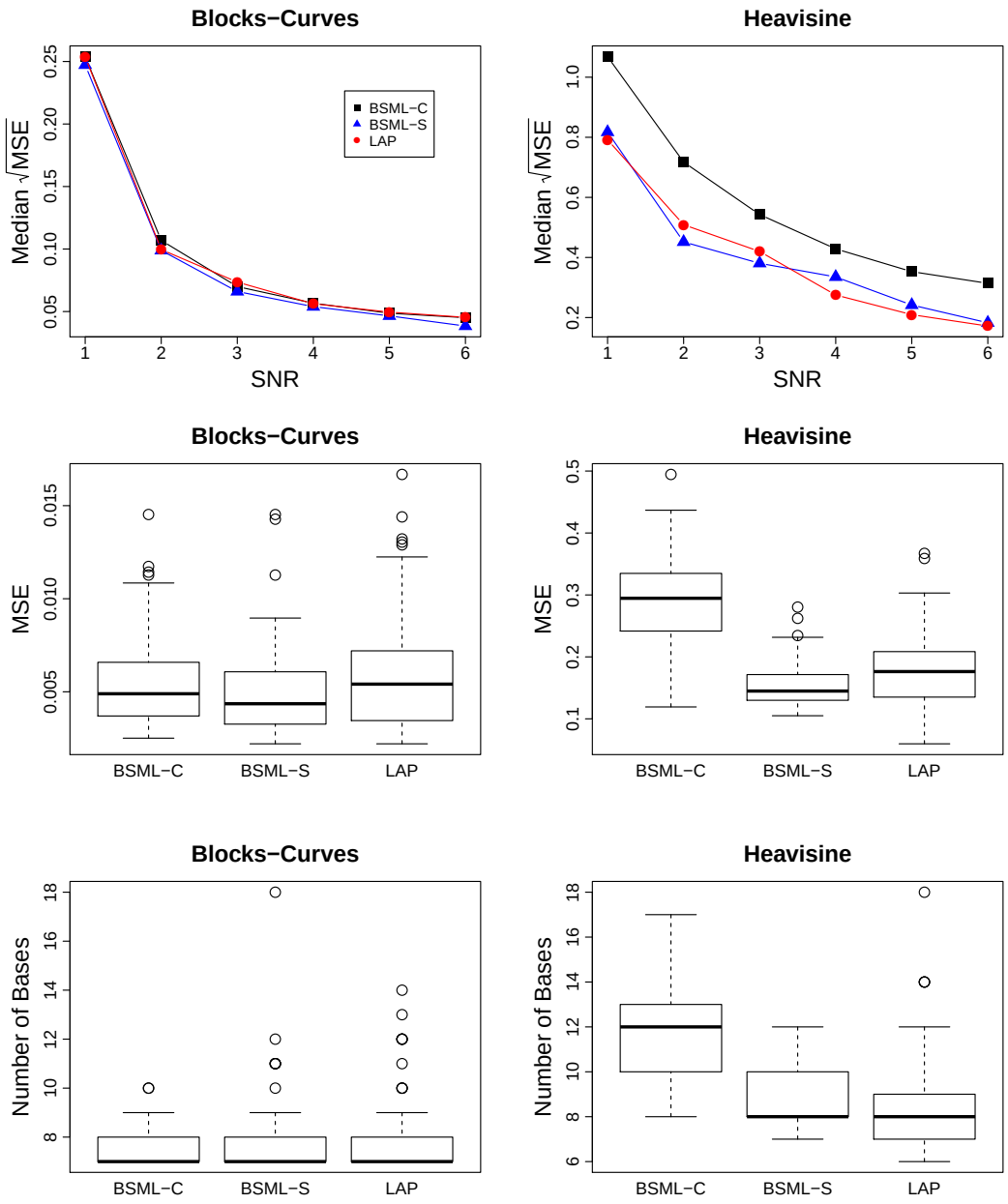


Figure 4.7: Simulation results for the Blocks-Curves and Heavisine examples with $n = 1024$. The top panel shows the median $\sqrt{\text{MSE}}$ across different SNR values for both examples, where the symbols used are: ■ for BSML-C, ▲ for BSML-S, and ● for the look-ahead procedure. The middle panel shows the boxplots of the MSE when SNR $= 3$. The bottom panel shows the number of basis functions selected when SNR $= 3$.

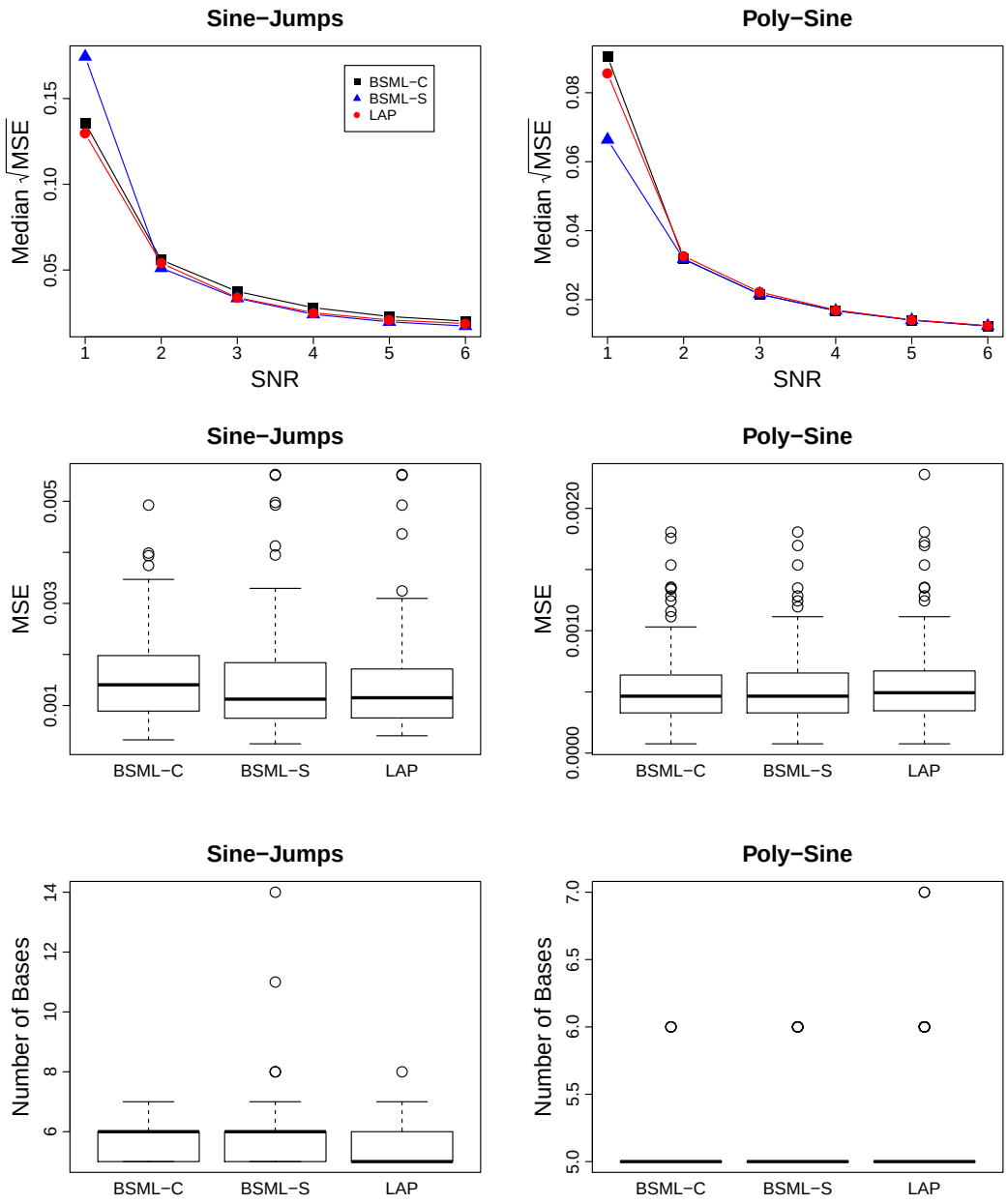


Figure 4.8: Simulation results for the Sine-Jumps and Poly-Sine examples with $n = 1024$. The top panel shows the median $\sqrt{\text{MSE}}$ across different SNR values for both examples, where the symbols used are: ■ for BSML-C, ▲ for BSML-S, and ● for the look-ahead procedure. The middle panel shows the boxplots of the MSE when SNR $= 3$. The bottom panel shows the number of basis functions selected when SNR $= 3$.

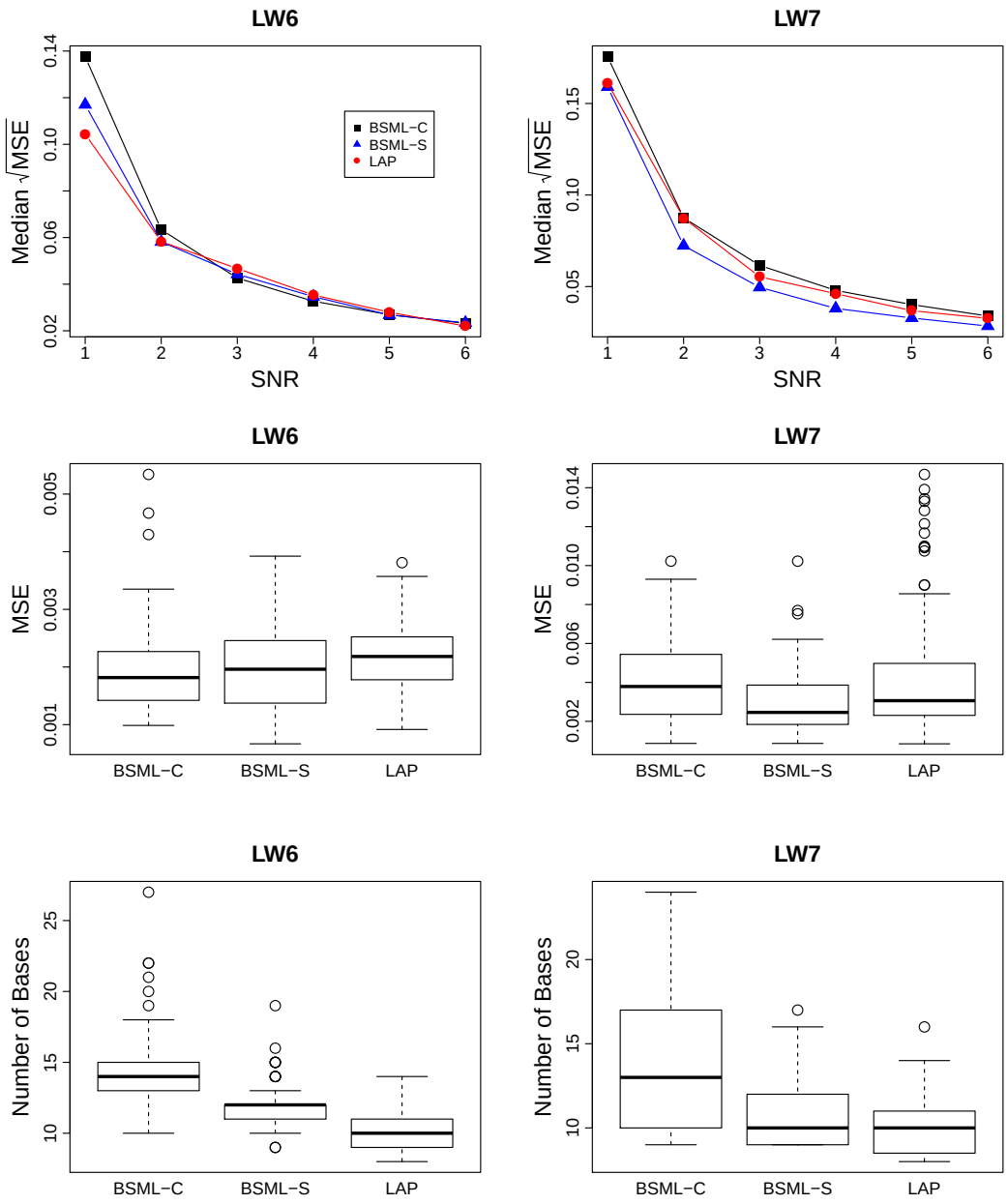


Figure 4.9: Simulation results for the LW6 and LW7 examples with $n = 1024$. The top panel shows the median $\sqrt{\text{MSE}}$ across different SNR values for both examples, where the symbols used are: ■ for BSML-C, ▲ for BSML-S, and ● for the look-ahead procedure. The middle panel shows the boxplots of the MSE when SNR $= 3$. The bottom panel shows the number of basis functions selected when SNR $= 3$.

Table 4.1: Average CPU time (in seconds) of fitting the Block-Curves and the Heavisine examples using BSML-C, BSML-S, and LAP for sample sizes $n = 256, 512, 1024$ with SNR = 3.

	Blocks-Curves			Heavisine		
	$n = 256$	$n = 512$	$n = 1024$	$n = 256$	$n = 512$	$n = 1024$
BSML-C	2.10	8.89	64.27	2.12	9.00	56.89
BSML-S	14.93	52.58	243.76	14.77	53.05	256.37
LAP	1.04	3.92	11.33	1.00	3.74	12.02

Table 4.2: Average CPU time (in seconds) of fitting the Sine-Jumps and the Poly-sine examples using BSML-C, BSML-S, and LAP for sample sizes $n = 256, 512, 1024$ with SNR = 3.

	Sine-Jumps			Poly-Sine		
	$n = 256$	$n = 512$	$n = 1024$	$n = 256$	$n = 512$	$n = 1024$
BSML-C	2.10	9.01	64.80	1.26	4.12	24.63
BSML-S	14.75	52.99	245.93	9.44	29.52	124.84
LAP	0.98	4.79	7.11	0.70	1.77	9.75

Table 4.3: Average CPU time (in seconds) of fitting the LW6 and LW7 examples using BSML-C, BSML-S, and LAP for sample sizes $n = 256, 512, 1024$ with SNR = 3.

	LW6			LW7		
	$n = 256$	$n = 512$	$n = 1024$	$n = 256$	$n = 512$	$n = 1024$
BSML-C	2.11	8.91	55.42	2.07	8.87	64.36
BSML-S	14.64	52.80	245.53	14.66	52.32	253.17
LAP	1.11	3.82	11.26	1.07	3.63	11.45

As we mentioned in Section 3.2, LAP only considers the pairs of basis functions of the form $\{\psi_l^-, \psi_{\cdot|l}^-\}$ for $l = 1, \ldots, L_-$, where $\psi_{\cdot|l}^-$ is selected given ψ_l^- already in the model. Alternatively, the selection scheme in (3.4) can be used, where all possible pairs of unselected basis functions are considered. This alternative procedure is referred to as LAP-2. The same 100 random samples of the six univariate examples with SNR = 3 and $n = 512$ are used to compare the performances of LAP and LAP-2. Figure 4.10 shows the boxplots of the MSE, and Table 4.4 lists the average CPU time for both procedures.

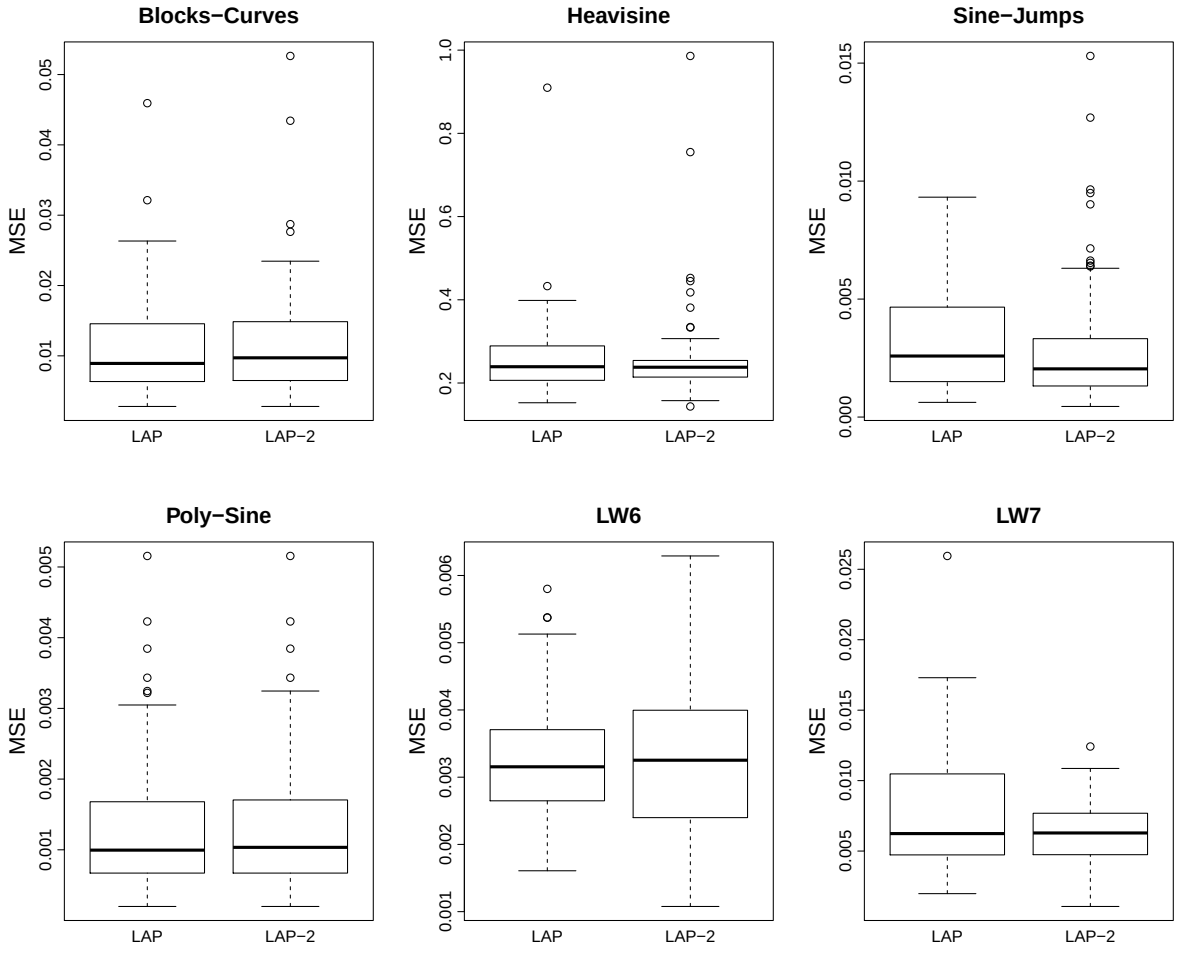


Figure 4.10: Boxplots of the MSE for fitting six univariate examples using LAP and LAP-2 with SNR = 3 and $n = 512$.

Table 4.4: Average CPU time (in seconds) of fitting each univariate example using LAP and LAP-2 with $n = 512$ and SNR $= 3$.

	Blocks-Curves	Heavisine	Sine-Jumps	LW6	LW7	Poly-Sine
LAP	3.92	3.74	4.79	3.82	3.63	1.77
LAP-2	124.33	64.7	142.58	127.31	109.03	13.65

We can see from Figure 4.10 and Table 4.4 that when all pairwise search scheme is used, there is no significant improvement in terms of MSE, but the computation time has increased dramatically. Therefore, we will use LAP, because it is less computationally demanding than LAP-2.

4.2 Bootstrap Confidence Intervals

We use the same Sine-Jumps and Blocks-Curves examples to evaluate the performance of the bootstrap confidence intervals from the LAP procedure, and compare them with that from the BSML procedures. For each example, we construct the 95% bootstrap confidence intervals under two settings: $n = 256, \text{SNR} = 4$ and $n = 512, \text{SNR} = 6$ with `bootrep`=200. All the other settings are exactly the same as in 4.1. Each simulation is repeated $n_r = 100$ times.

The pointwise coverage probability (PCP) and average coverage probability (ACP) are calculated to evaluate the performance of the confidence intervals. For any n arbitrary points $\{\boldsymbol{x}_1, \ldots, \boldsymbol{x}_n\}$ in the domain, let $f(\boldsymbol{x}_i)$ be the true function evaluated at $\boldsymbol{x}_i$. For the r-th simulated data set $\big\{(\boldsymbol{x}_1, y_{1,r}), \ldots, (\boldsymbol{x}_n, y_{n,r})\big\}$, where $r = 1, \ldots, n_r$, let $\hat{f}^{(r)}_{\alpha/2}(\boldsymbol{x}_i)$ and $\hat{f}^{(r)}_{1-\alpha/2}(\boldsymbol{x}_i)$ be the lower and upper bound of the $100(1-\alpha)\%$ bootstrap confidence interval at $\boldsymbol{x}_i$. Then the pointwise coverage probability (PCP) at $\boldsymbol{x}_i$ is defined as

$$\text{PCP}_i = \frac{1}{n_r} \sum_{r=1}^{n_r} I\Big(f(\boldsymbol{x}_i) \in \big[\hat{f}^{(r)}_{\alpha/2}(\boldsymbol{x}_i), \hat{f}^{(r)}_{1-\alpha/2}(\boldsymbol{x}_i)\big]\Big), \tag{4.1}$$

where $I(\cdot)$ is the indicator function. The average coverage probability (ACP) is computed for each $100(1-\alpha)\%$ bootstrap confidence interval across the function, defined as

$$\text{ACP}_r = \frac{1}{n} \sum_{i=1}^{n} I\Big(f(\boldsymbol{x}_i) \in \big[\hat{f}^{(r)}_{\alpha/2}(\boldsymbol{x}_i), \hat{f}^{(r)}_{1-\alpha/2}(\boldsymbol{x}_i)\big]\Big). \tag{4.2}$$

Besides coverage probabilities, we are also interested in the width of a confidence interval, since given the same coverage probability a narrower confidence interval provides more precise information about the true function than a wider one. We define the pointwise average interval width (PAIW) at each design point $\boldsymbol{x}_i$ as follows, where the

average is across the results for the n_r simulated data sets:

$$\text{PAIW}_i = \frac{1}{n_r}\sum_{r=1}^{n_r}\left(\hat{f}^{(r)}_{1-\alpha/2}(\boldsymbol{x}_i) - \hat{f}^{(r)}_{\alpha/2}(\boldsymbol{x}_i)\right). \tag{4.3}$$

For the Blocks-Curves and Sine-Jumps examples, Figures 4.11 to 4.14 show the comparisons of the PCP, ACP, and PAIW based on the 95% bootstrap confidence intervals from the LAP, BSML-C, and BSML-S procedures for $n_r = 100$. For all settings the BSML-S confidence intervals gave the lowest median ACP and virtually always the lowest PCP values. The median ACP for BSML-S confidence intervals for each of the four simulation cases considered is below the nominal value 0.95. The BSML-C confidence intervals and LAP confidence intervals have similar performances, and the median ACP for both of them are above the nominal value for all settings. The advantage of the LAP confidence intervals is that in general they are pointwise narrower than the BSML-C confidence intervals. For the Sine-Jumps example the LAP confidence intervals have shorter widths almost uniformly than the BSML-C confidence intervals, except at the places near 0.25 and 0.5 where the Sine-Jumps function is discontinuous.

We can also see that coverage probabilities depend on the true function, as well as the sample size and signal-to-noise ratio. Although the median ACP for all LAP confidence interval is above the nominal value of 0.95, the pointwise coverage probabilities can still be below 0.95. For the Blocks-Curves example when $n = 512$ and SNR $= 6$, the PCP is much lower than 0.95 at the design points near 0.7 where the Blocks-Curves function has a big jump. Therefore, same as the conclusions in Wang and Wahba (1995), the bootstrap confidence intervals from the LAP procedure should be interpreted as across the curve, instead of pointwise. As we mentioned earlier, the bias of the estimates tends to be large at places where f has sharp curvature or sudden jumps. Our simulations support this claim, and we found that the confidence intervals tend to be very wide at

these places as well.

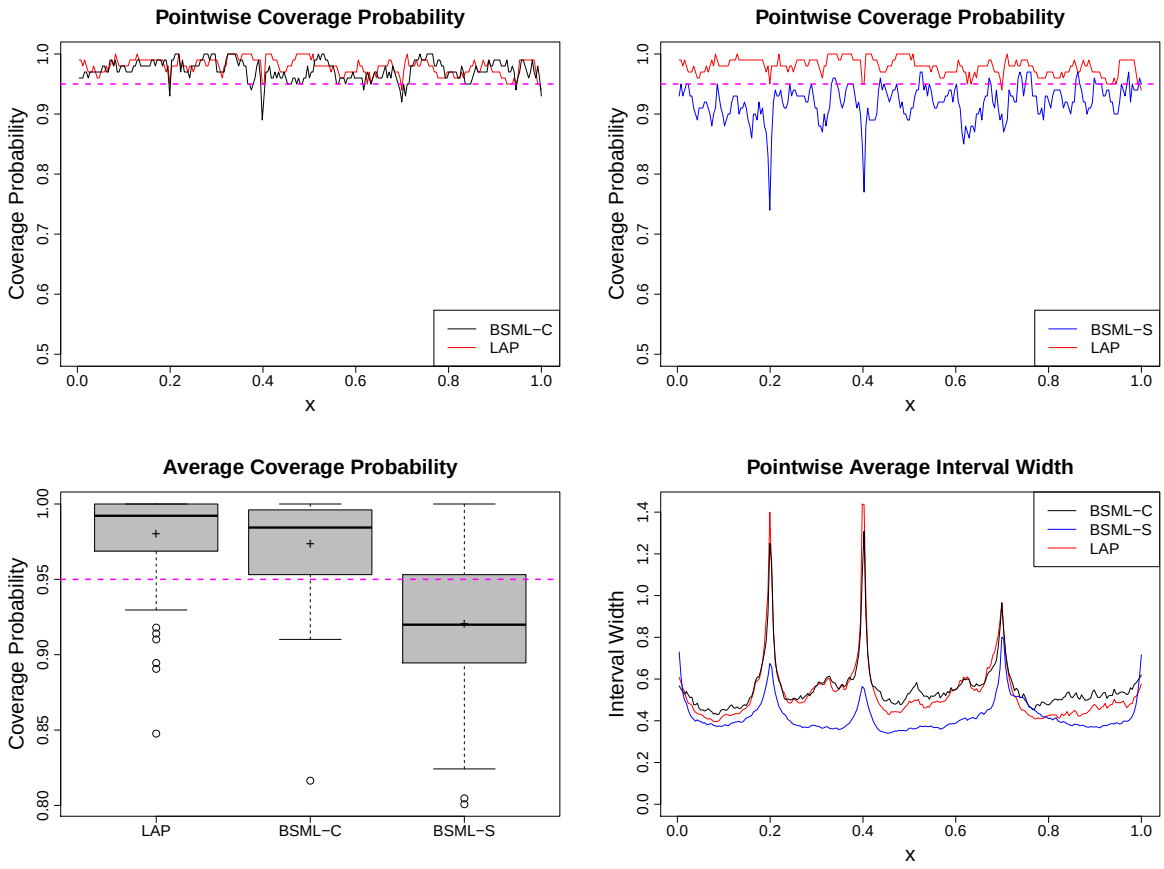


Figure 4.11: PCP, ACP, PAIW of the 95% bootstrap confidence intervals from the LAP, BSML-C, and BSML-S procedures using the Blocks-Curves example with SNR $= 4$ and $n = 256$. Top left: pointwise coverage probability comparisons between the LAP and BSML-C confidence intervals. Top right: pointwise coverage probability comparisons between the LAP and BSML-S confidence intervals. Bottom left: boxplots of the average coverage probabilities of the LAP, BSML-C, and BSML-S confidence intervals. Bottom right: pointwise average interval width of the LAP, BSML-C, and BSML-S confidence intervals. In the PCP and PAIW plots, red solid curve represents LAP, black solid curve represents BSML-C, and blue solid curve represents BSML-S. In the three coverage probability plots, the magenta dashed line represents the nominal coverage probability of 0.95. The + symbols in the ACP panel show the means for the three methods.

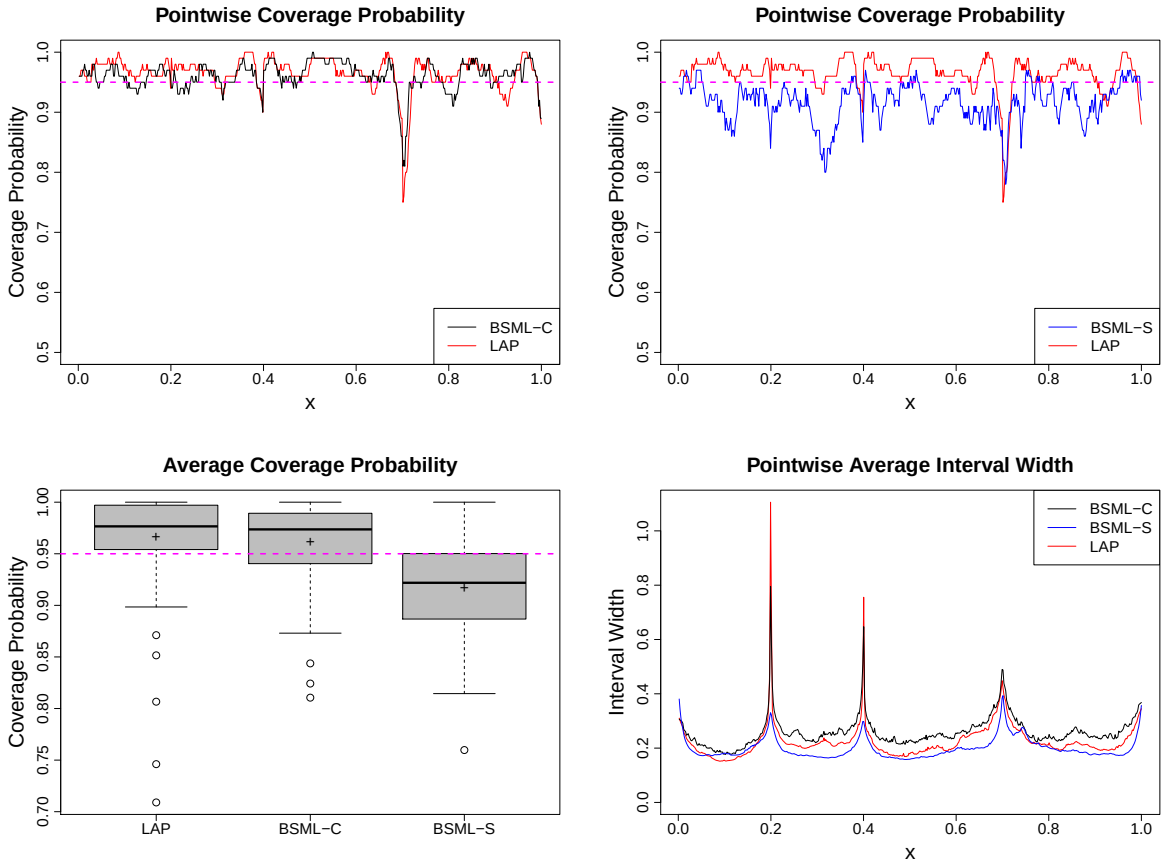


Figure 4.12: PCP, ACP, PAIW of the 95% bootstrap confidence intervals from LAP, BSML-C, and BSML-S procedures using the Blocks-Curves example with SNR = 6 and $n = 512$. Top left: pointwise coverage probability comparisons between the LAP and BSML-C confidence intervals. Top right: pointwise coverage probability comparisons between the LAP and BSML-S confidence intervals. Bottom left: boxplots of the average coverage probabilities of the LAP, BSML-C, and BSML-S confidence intervals. Bottom right: pointwise average interval width of the LAP, BSML-C, and BSML-S confidence intervals. In the PCP and PAIW plots, red solid curve represents LAP, black solid curve represents BSML-C, and blue solid curve represents BSML-S. In the three coverage probability plots, the magenta dashed line represents the nominal coverage probability of 0.95. The + symbols in the ACP panel show the means for the three methods.

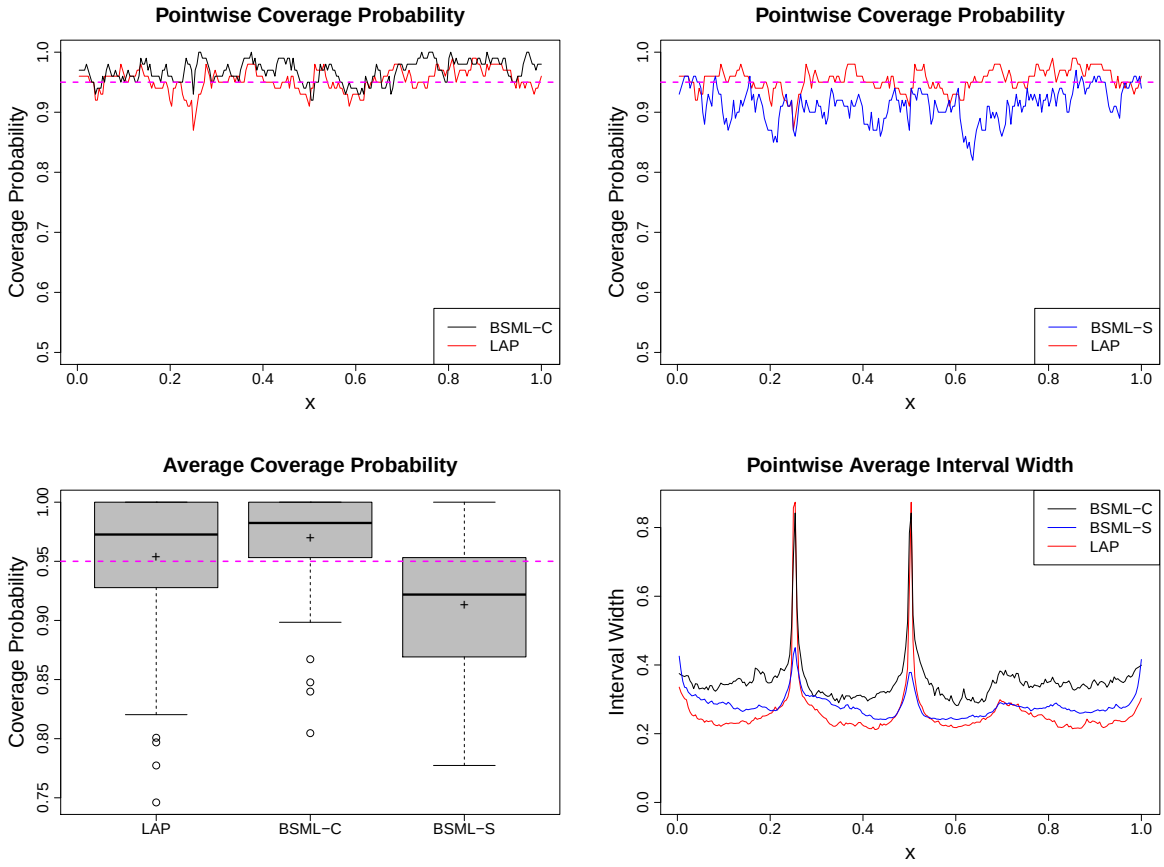


Figure 4.13: PCP, ACP, PAIW of the 95% bootstrap confidence intervals from LAP, BSML-C, and BSML-S procedures using the Sine-Jumps example with SNR $= 4$ and $n = 256$. Top left: pointwise coverage probability comparisons between the LAP and BSML-C confidence intervals. Top right: pointwise coverage probability comparisons between the LAP and BSML-S confidence intervals. Bottom left: boxplots of the average coverage probabilities of the LAP, BSML-C, and BSML-S confidence intervals. Bottom right: pointwise average interval width of the LAP, BSML-C, and BSML-S confidence intervals. In the PCP and PAIW plots, red solid curve represents LAP, black solid curve represents BSML-C, and blue solid curve represents BSML-S. In the three coverage probability plots, the magenta dashed line represents the nominal coverage probability of 0.95. The + symbols in the ACP panel show the means for the three methods.

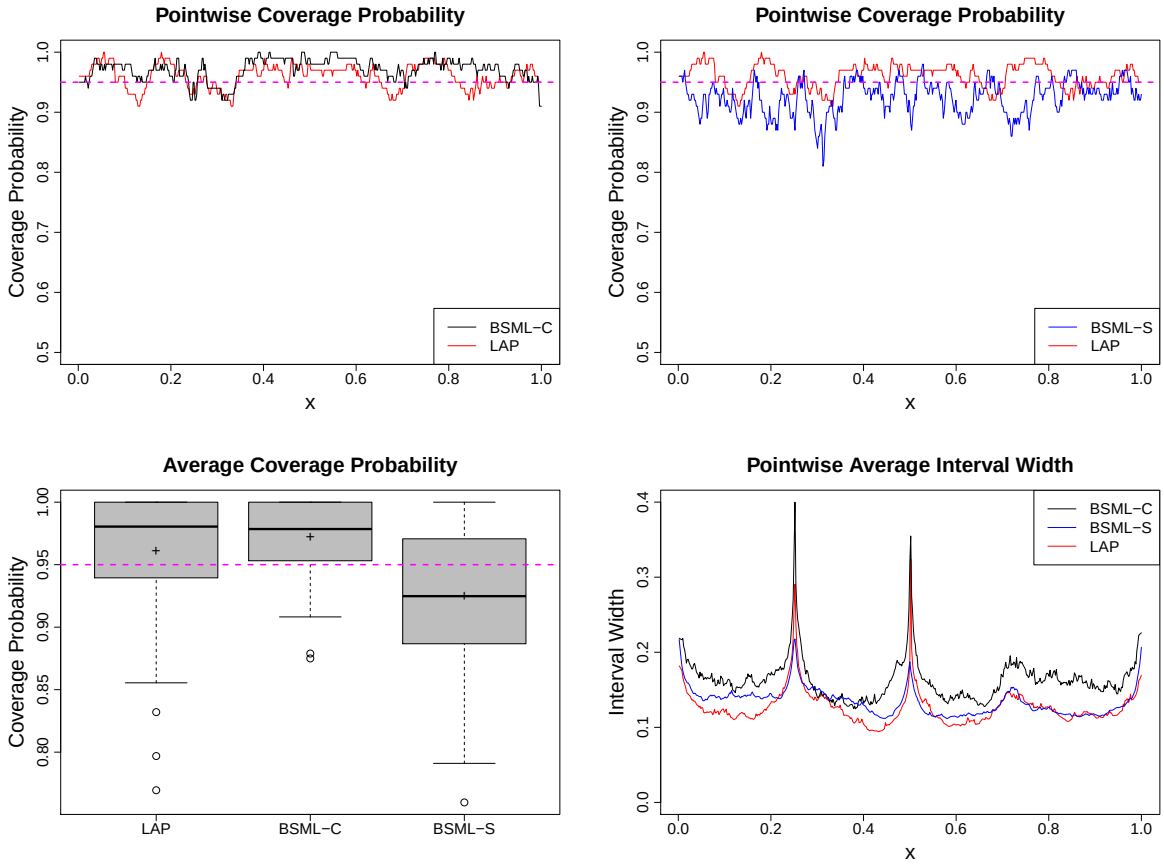


Figure 4.14: PCP, ACP, PAIW of the 95% bootstrap confidence intervals from LAP, BSML-C, and BSML-S procedures using the Sine-Jumps example with SNR $= 6$ and $n = 512$. Top left: pointwise coverage probability comparisons between the LAP and BSML-C confidence intervals. Top right: pointwise coverage probability comparisons between the LAP and BSML-S confidence intervals. Bottom left: boxplots of the average coverage probabilities of the LAP, BSML-C, and BSML-S confidence intervals. Bottom right: pointwise average interval width of the LAP, BSML-C, and BSML-S confidence intervals. In the PCP and PAIW plots, red solid curve represents LAP, black solid curve represents BSML-C, and blue solid curve represents BSML-S. In the three coverage probability plots, the magenta dashed line represents the nominal coverage probability of 0.95. The + symbols in the ACP panel show the means for the three methods.

4.3 Estimation of Multivariate Functions

4.3.1 Additive Model Examples

Now consider the bivariate nonparametric regression model

$$y_i = f_1(x_{i1}) + f_2(x_{i2}) + \varepsilon_i, \quad i = 1, \ldots, n, \tag{4.4}$$

where $x_{i1} \in [0,1]$ and $x_{i2} \in [0,1]$ are the observations of two explanatory variables for observation y_i, and the independent random errors $\boldsymbol{\varepsilon} = (\varepsilon_1, \ldots, \varepsilon_n)$ have mean 0 and constant variance σ^2. We choose from the same six univariate functions in Table 2.1 to construct three combinations, which are shown in Table 4.5. Table 4.6 contains the basis libraries used in BSML-C, BSML-S, and LAP for each example.

Table 4.5: Test functions for additive model simulations.

Function Name	True Function $f(\boldsymbol{x})$	$SD(f)$
Sine-Jumps + LW7	$\sin(2\pi x_1) - 1_{(0.5,1]}(x_1) + 1_{(0.25,1]}(x_1)$ $+(4x_2 - 2) + 2\exp[-256(x_2 - 0.5)^2]$	1.6141
Blocks-Curves + Sine-Jumps	$1_{[0.2,0.4)}(x_1) + \exp[(x_1 + 0.5)^2]1_{[0.4,0.7)}(x_1) - x_1^{10}1_{[0.7,1]}(x_1)$ $+\sin(2\pi x_1) - 1_{(0.5,1]}(x_1) + 1_{(0.25,1]}(x_1)$	1.7702
Poly-Sine + LW6	$\sin(16\pi x_1) - 8(x_1 - 0.5)^2 + 8(x_1 - 0.5)^3 1_{(0.5,1]}(x_1)$ $+\sin[2(4x_2 - 2)] + 2\exp[-256(x_2 - 0.5)^2]$	1.1866

In the simulation, the design points (x_{i1}, x_{i2}) for $i = 1, \ldots, n$ are uniformly sampled from $[0,1] \times [0,1]$, where the sample size is $n \in \{256, 512, 1024\}$. The error variance σ^2 is chosen such that SNR $\in \{1, 2, 3, 4, 5, 6\}$. The knots used in all libraries are grid points $\{0.06 + 0.02j, j = 1, 2, \ldots, 44\}$. For the BSML and LAP procedures, the maximal number of bases M is fixed at 30. For LAP, the IDF spaces are $S_i = \{1, 2, 3\}$ for $i = 1, 2, 3, 4$, and M_d is set to be $\prod_{i=1}^{4} |S_i| = 3^4 = 81$. The 10-fold cross validation is used to select

Table 4.6: Basis libraries used in BSML-C, BSML-S, and LAP for each additive model example. Notations of basis libraries can be found in Table 1.1.

Function Name	Basis Libraries				
	$\mathcal{L}_0$	$\mathcal{L}_1$	$\mathcal{L}_2$	$\mathcal{L}_3$	$\mathcal{L}_4$
Sine-Jumps + LW7	$\{1, x_1, x_2\}$	$\mathcal{P}_2(x_1)$	$\mathcal{T}_0(x_1)$	$\mathcal{C}_2(x_2)$	$\mathcal{T}_2(x_2)$
Blocks-Curves + Sine-Jumps	$\{1, x_1, x_2\}$	$\mathcal{C}_2(x_1)$	$\mathcal{T}_0(x_1)$	$\mathcal{P}_2(x_2)$	$\mathcal{T}_0(x_2)$
Poly-Sine + LW6	$\{1, x_1, x_2\}$	$\mathcal{F}_{8,25}(x_1)$	$\mathcal{T}_2(x_1)$	$\mathcal{C}_2(x_2)$	$\mathcal{T}_2(x_2)$

the final model in the look-ahead procedure.

We also applied the SS-ANOVA model, the COSSO method, and the MARS procedure to these three examples. We used the function `ssr` in the `R` package `assist` (Wang and Ke (2015)) to fit the SS-ANOVA models. For the Blocks-Curves + Sine-Jumps example, we assume $f \in W_2^2[0,1] \otimes W_2^2(per)$. For the Sine-Jumps + LW7 example, we assume $f \in W_2^2(per) \otimes W_2^2[0,1]$. For the Poly-Sine + LW6 example, we assume $f \in W_2^2[0,1] \otimes W_2^2[0,1]$. For all SS-ANOVA models, only the main effects are included. For the COSSO method, we used the function `cosso` in the `R` package `cosso` (Zhang and Lin (2013)), with 10-fold cross validation for the tuning parameter selection. For the MARS procedure, we used the function `mars` in the `R` package `mda` (Hastie et al. (2015)), with `degree` = `1` which means no interaction term is included. The IDF used in `mars` is fixed at 3, and the maximum number of model terms `nk` is fixed at 30. The simulation results for BSML-C, BSML-S, LAP, SS-ANOVA, COSSO, and MARS based on 100 random samples are shown in Figures 4.15 – 4.17. The average CPU time for each sample size is listed in Table 4.7.

Based on the results, MARS did not work well when SNR ≥ 2 for all three examples. For COSSO, it did not work well for the Poly-Sine + LW6 example. When SNR ≥ 4, LAP has the best performance for the Blocks-Curves + Sine-Jumps and Polysine + LW6 examples across all sample sizes, and also for the Sine-Jumps + LW7 example when

the sample size is 256 or 512. On average LAP has selected fewer basis functions than BSML-C and BSML-S. SS-ANOVA has comparable performance as the BSML and LAP procedures, but in terms of computation time it has disadvantage when the sample size is big, e.g., when $n = 1024$. LAP and BSML-C have similar computation speed, both are much faster than BSML-S.

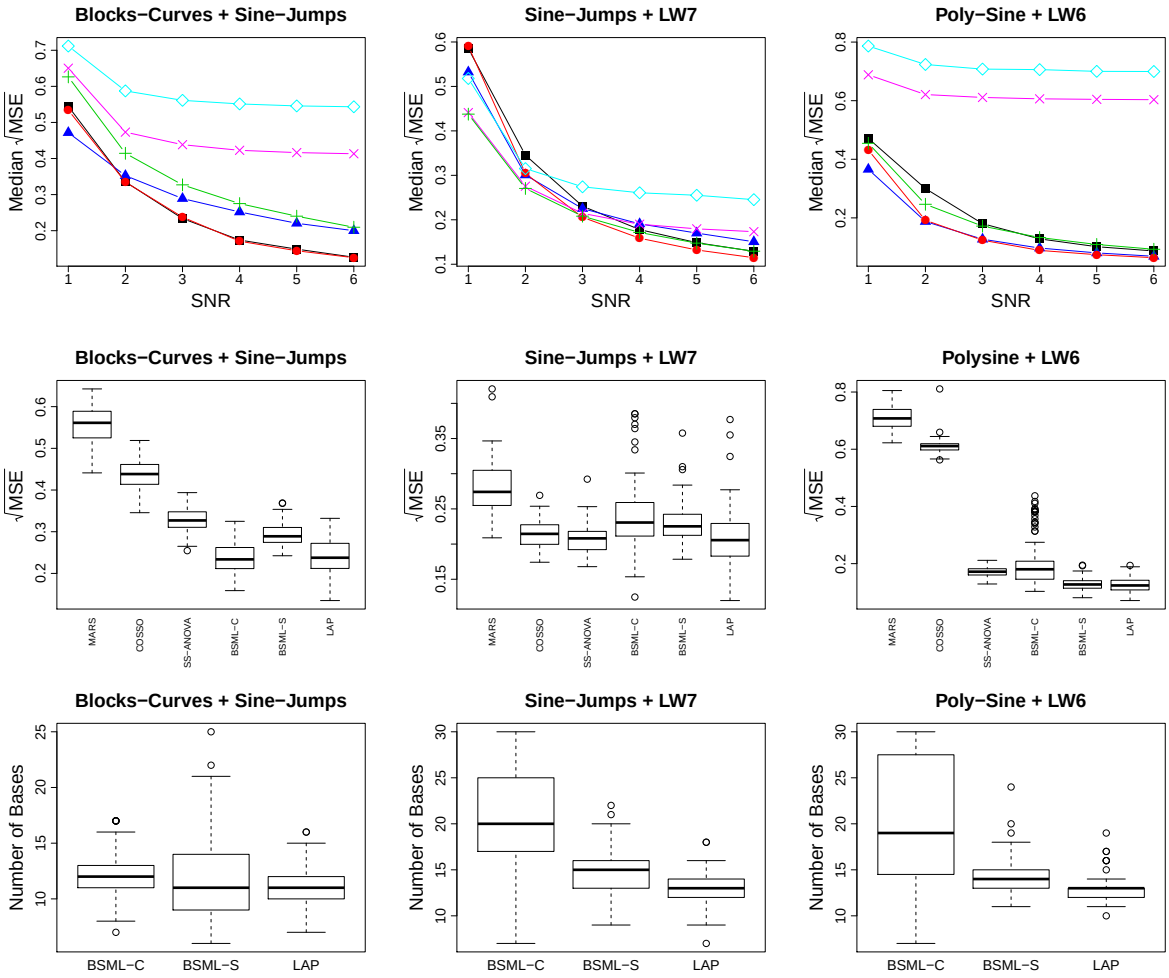


Figure 4.15: Simulation results for three additive model examples with $n = 256$. The top panel shows the median $\sqrt{\text{MSE}}$ across different SNR values for each example, where the symbols used are: ■ for BSML-C, ▲ for BSML-S, ● for LAP, + for SS-ANOVA, × for COSSO, and ◇ for MARS. The middle panel shows the boxplots of the $\sqrt{\text{MSE}}$ when SNR = 3. The bottom panel shows the number of basis functions selected by BSML-C, BSML-S, and LAP when SNR = 3.

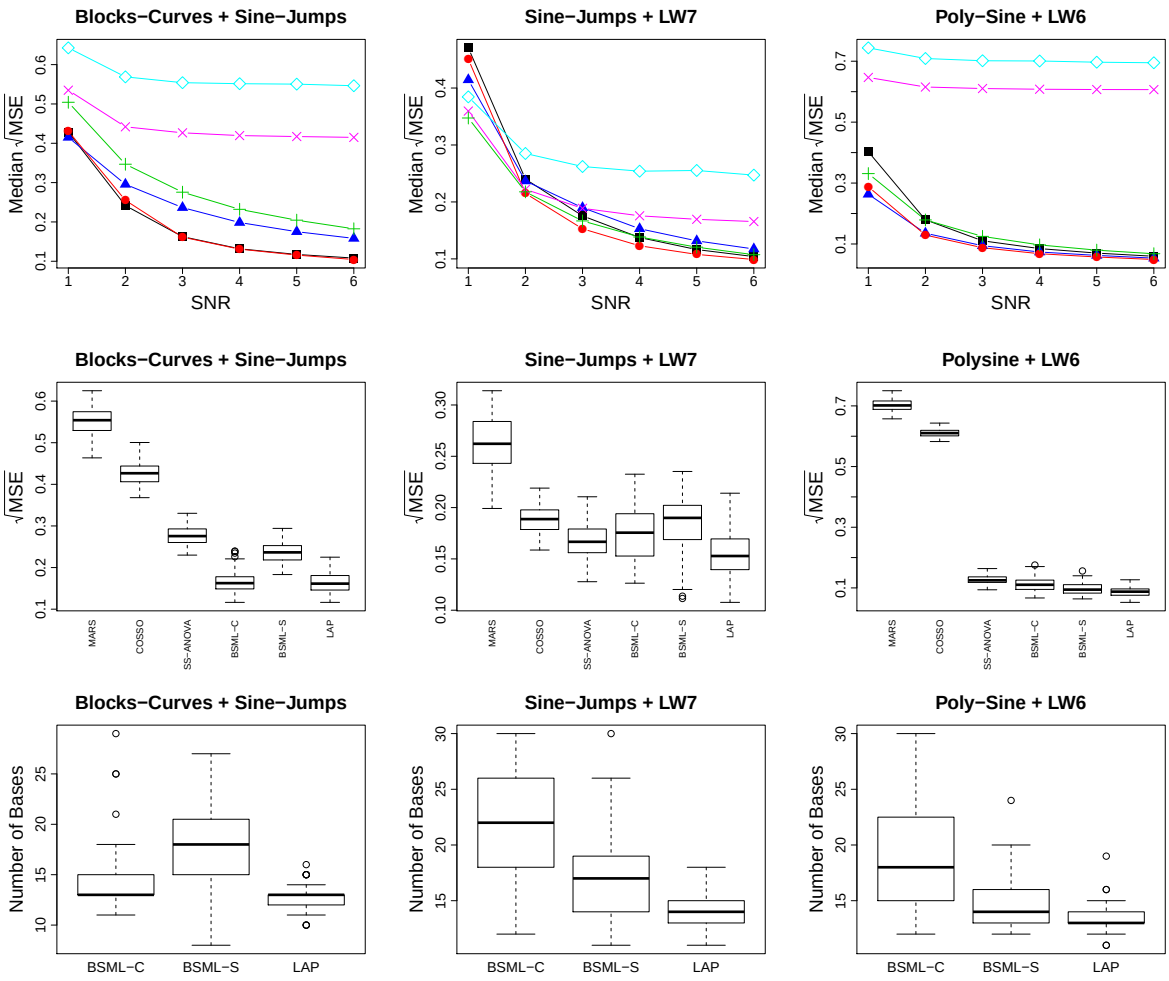


Figure 4.16: Simulation results for three additive model examples with $n = 512$. The top panel shows the median $\sqrt{\text{MSE}}$ across different SNR values for each example, where the symbols used are: ■ for BSML-C, ▲ for BSML-S, ● for LAP, + for SS-ANOVA, × for COSSO, and ◇ for MARS. The middle panel shows the boxplots of the $\sqrt{\text{MSE}}$ when SNR $= 3$. The bottom panel shows the number of basis functions selected by BSML-C, BSML-S, and LAP when SNR $= 3$.

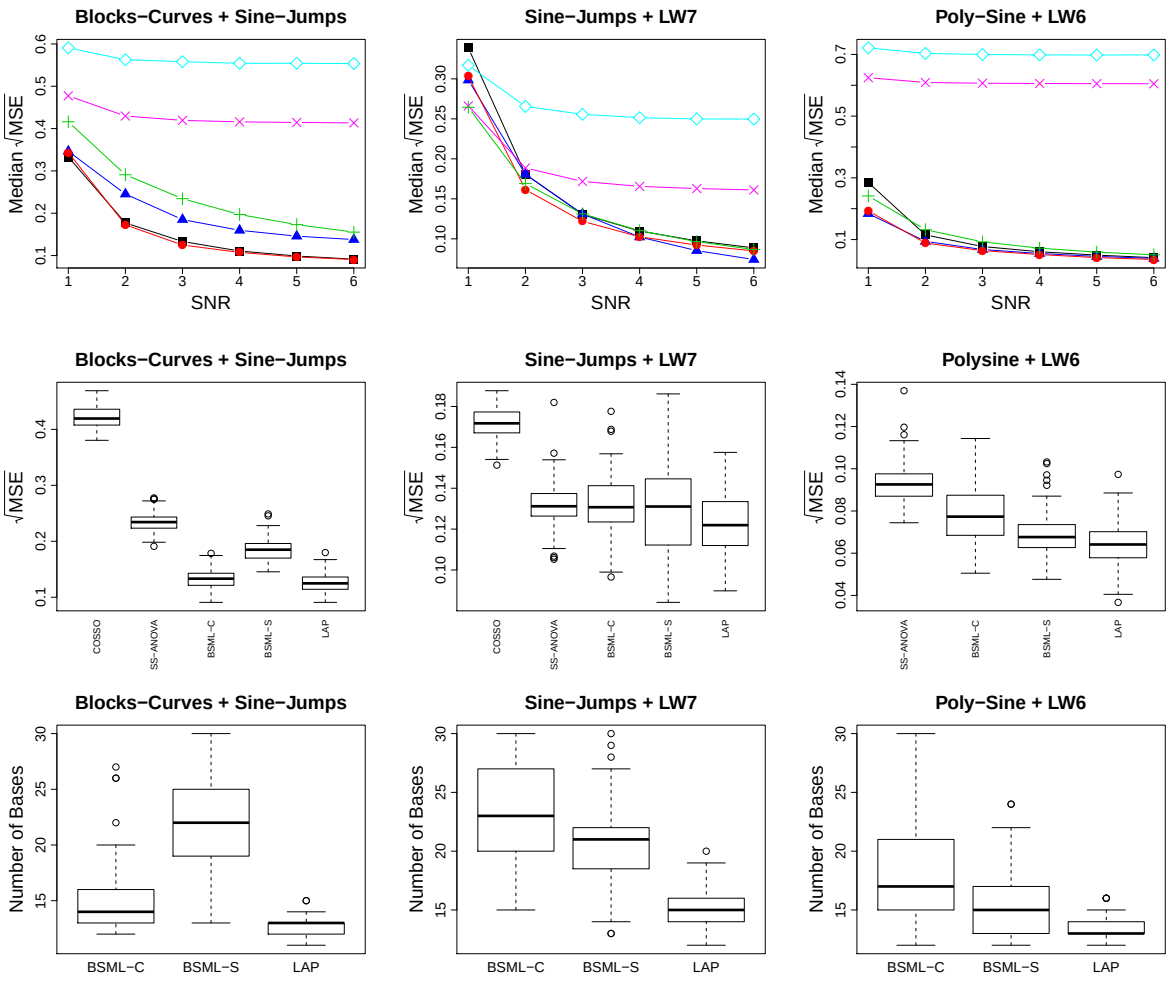


Figure 4.17: Simulation results for three additive model examples with $n = 1024$. The top panel shows the median $\sqrt{\text{MSE}}$ across different SNR values for each example, where the symbols used are: ■ for BSML-C, ▲ for BSML-S, ● for LAP, + for SS-ANOVA, × for COSSO, and ◇ for MARS. The middle panel shows the boxplots of the $\sqrt{\text{MSE}}$ when SNR $= 3$. The results of fitting all three examples with MARS and the result of fitting the Polysine + LW6 example with COSSO are not shown here because they are much worse compared to the results from the BSML and LAP procedures. The bottom panel shows the number of basis functions selected by BSML-C, BSML-S, and LAP when SNR $= 3$.

Table 4.7: Average CPU time (in seconds) of fitting each additive model example using BSML-C, BSML-S, LAP, SS-ANOVA, COSSO, MARS for sample sizes $n = 256, 512, 1024$ with SNR $= 3$.

	Blocks-Curves + Sine-Jumps			Sine-Jumps + LW7		
	$n = 256$	$n = 512$	$n = 1024$	$n = 256$	$n = 512$	$n = 1024$
BSML-C	0.89	1.73	3.51	1.04	1.35	3.29
BSML-S	8.23	16.26	30.82	8.16	22.23	37.46
LAP	0.65	1.37	2.04	1.09	1.86	2.85
SS-ANOVA	0.32	1.66	9.42	0.18	1.00	6.87
COSSO	0.39	0.87	3.11	0.27	0.86	2.61
MARS	0.05	0.06	0.05	0.02	0.02	0.02

	Poly-Sine + LW6		
	$n = 256$	$n = 512$	$n = 1024$
BSML-C	1.04	1.39	3.33
BSML-S	8.36	22.80	39.34
LAP	0.81	1.18	1.56
SS-ANOVA	0.20	1.05	6.00
COSSO	0.30	0.94	2.88
MARS	0.02	0.02	0.03

4.3.2 Models with Interactions

Consider the bivariate nonparametric regression model

$$y_i = f_{12}(x_{i1}, x_{i2}) + \varepsilon_i, \quad i = 1, \ldots, n, \tag{4.5}$$

where $x_{i1} \in [0,1]$ and $x_{i2} \in [0,1]$ are the observations of two explanatory variables for observation y_i, and the independent random errors $\boldsymbol{\varepsilon} = (\varepsilon_1, \ldots, \varepsilon_n)$ have mean 0 and constant variance σ^2. We constructed three examples of such function f, which are shown in Table 4.8. Table 4.9 and 4.10 contain the basis libraries used in the BSML and LAP procedures for each example. Here the knots used are $z_j = 0.06 + 0.02j,\ j = 1, \ldots, 44$. For all three examples, $\mathcal{L}_1$ to $\mathcal{L}_5$ contain similar spline representers used in SS-ANOVA model with the model space $W_2^m[0,1] \otimes W_2^m[0,1]$. $\mathcal{L}_6$ for the SJ·LW7 and BC·LW6 examples contains 2d step functions in x_1, which are used to capture the jumps in the true functions.

Table 4.8: Test functions for models with interactions.

Function Name	True Function $f(\boldsymbol{x})$	$SD(f)$
SJ·LW7	$2\big[0.5\sin(2\pi x_1) - 1_{(0.5,1]}(x_1) + 1_{(0.24,1]}(x_1)\big]$ $+4x_2 + 6(x_1 - 0.5)\exp[-256(x_2 - 0.5)^2]$	1.7930
BC·LW6	$1_{[0.2,0.4)}(x_1) + \exp[(x_1 + 0.5)^2]1_{[0.4,0.7)}(x_1) - x_1^{10}1_{[0.7,1]}(x_1)$ $+\sin[2(4x_2 - 2)] + 6(x_1 - 0.5)\exp[-256(x_2 - 0.5)^2]$	1.6671
LW6·LW7	$\sin[2(4x_1 - 2)] + 6(x_2 - 0.5)\exp[-256(x_1 - 0.5)^2]$ $+4(x_2 - 0.5) + 2\exp[-256(x_2 - 0.5)^2]$	1.6483

In the simulation, the design points (x_{i1}, x_{i2}) for $i = 1, \ldots, n$ are uniformly sampled from $[0,1] \times [0,1]$ for all examples. The sample size used is $n \in \{256, 512, 1024\}$. The error variance σ^2 is chosen such that SNR $\in \{1, 2, 3, 4, 5, 6\}$. For the BSML and LAP procedures, the maximal number of bases M is fixed at 40. For the look-ahead procedure,

the IDF spaces are $S_i = \{1, 2, 3\}$ for each library $\mathcal{L}_i$. M_d is set to be $\prod_{i=1}^{6} |S_i| = 3^6 = 729$ for the SJ·LW7 and BC·LW6 examples, and $\prod_{i=1}^{5} |S_i| = 3^5 = 243$ for the LW6·LW7 example. The 10-fold cross validation is used to select the final model in the look-ahead procedure.

Table 4.9: Basis libraries for the SJ·LW7 and BC·LW6 examples.

Basis libraries for the SJ·LW7 and BC·LW6 examples
$\mathcal{L}_0 = \{1, x_1 - 0.5, x_2 - 0.5, (x_1 - 0.5)(x_2 - 0.5)\}$
$\mathcal{L}_1 = \mathcal{C}_2(x_1) = \left\{ \{R_2(x_{i1}, z_1)\}_{i=1}^n, \ldots, \{R_2(x_{i1}, z_q)\}_{i=1}^n \right\}$
$\mathcal{L}_2 = \mathcal{C}_2(x_2) = \left\{ \{R_2(x_{i2}, z_1)\}_{i=1}^n, \ldots, \{R_2(x_{i2}, z_q)\}_{i=1}^n \right\}$
$\mathcal{L}_3 = \left\{ \{(z_1 - 0.5)(x_{i1} - 0.5)R_2(x_{i2}, z_1)\}_{i=1}^n, \ldots, \{(z_q - 0.5)(x_{i1} - 0.5)R_2(x_{i2}, z_q)\}_{i=1}^n \right\}$
$\mathcal{L}_4 = \left\{ \{(z_2 - 0.5)(x_{i2} - 0.5)R_2(x_{i1}, z_1)\}_{i=1}^n, \ldots, \{(z_q - 0.5)(x_{i2} - 0.5)R_2(x_{i1}, z_q)\}_{i=1}^n \right\}$
$\mathcal{L}_5 = \left\{ \{R_2(x_{i1}, z_1)R_2(x_{i2}, z_1)\}_{i=1}^n, \ldots, \{R_2(x_{i1}, z_q)R_2(x_{i2}, z_q)\}_{i=1}^n \right\}$
$\mathcal{L}_6 = \mathcal{T}_0(x_1) = \left\{ \{1_{(z_1,1]}(x_{i1})\}_{i=1}^n, \ldots, \{1_{(z_q,1]}(x_{i1})\}_{i=1}^n \right\}$

Table 4.10: Basis libraries for the LW6·LW7 example.

Basis libraries for the LW6·LW7 example
$\mathcal{L}_0 = \{1, x_1 - 0.5, x_2 - 0.5, (x_1 - 0.5)(x_2 - 0.5)\}$
$\mathcal{L}_1 = \mathcal{C}_2(x_1) = \left\{ \{R_2(x_{i1}, z_1)\}_{i=1}^n, \ldots, \{R_2(x_{i1}, z_q)\}_{i=1}^n \right\}$
$\mathcal{L}_2 = \mathcal{C}_2(x_2) = \left\{ \{R_2(x_{i2}, z_1)\}_{i=1}^n, \ldots, \{R_2(x_{i2}, z_q)\}_{i=1}^n \right\}$
$\mathcal{L}_3 = \left\{ \{(z_1 - 0.5)(x_{i1} - 0.5)R_2(x_{i2}, z_1)\}_{i=1}^n, \ldots, \{(z_q - 0.5)(x_{i1} - 0.5)R_2(x_{i2}, z_q)\}_{i=1}^n \right\}$
$\mathcal{L}_4 = \left\{ \{(z_2 - 0.5)(x_{i2} - 0.5)R_2(x_{i1}, z_1)\}_{i=1}^n, \ldots, \{(z_q - 0.5)(x_{i2} - 0.5)R_2(x_{i1}, z_q)\}_{i=1}^n \right\}$
$\mathcal{L}_5 = \left\{ \{R_2(x_{i1}, z_1)R_2(x_{i2}, z_1)\}_{i=1}^n, \ldots, \{R_2(x_{i1}, z_q)R_2(x_{i2}, z_q)\}_{i=1}^n \right\}$

We also applied the SS-ANOVA model and the MARS procedure to these three examples. For all examples, SS-ANOVA model assumes $f \in W_2^2[0,1] \otimes W_2^2[0,1]$. For the MARS procedure, we set `degree = 2` in `mars` which means we allow pairwise interactions terms in the model. The IDF used is fixed at 3, and the maximum number of model terms `nk` is fixed at 40. The simulation results for BSML-C, BSML-S, LAP, SS-ANOVA, and MARS based on 100 random samples are shown in Figure 4.18 – 4.20. The average CPU time for each sample size is listed in Table 4.11.

Based on the results, for both the SJ·LW7 and BC·LW6 examples LAP has the best performance across all sample sizes when SNR ≥ 3. For these two examples, it makes sense that the LAP and BSML procedures outperform SS-ANOVA, since they can select the step functions from $\mathcal{L}_6$ to fit the jumps perfectly while SS-ANOVA can only smooth out the jumps using the cubic spline representers. For the LW6·LW7 example, however, because the true function is continuous and can be approximated well by using only the cubic spline representers, SS-ANOVA has better performance than the LAP and BSML procedures. For all three examples, on average LAP selected fewer basis functions than BSML-C and BSML-S. In terms of the computation speed, LAP is not as fast as BSML-C, but is still at least 3 times faster than BSML-S.

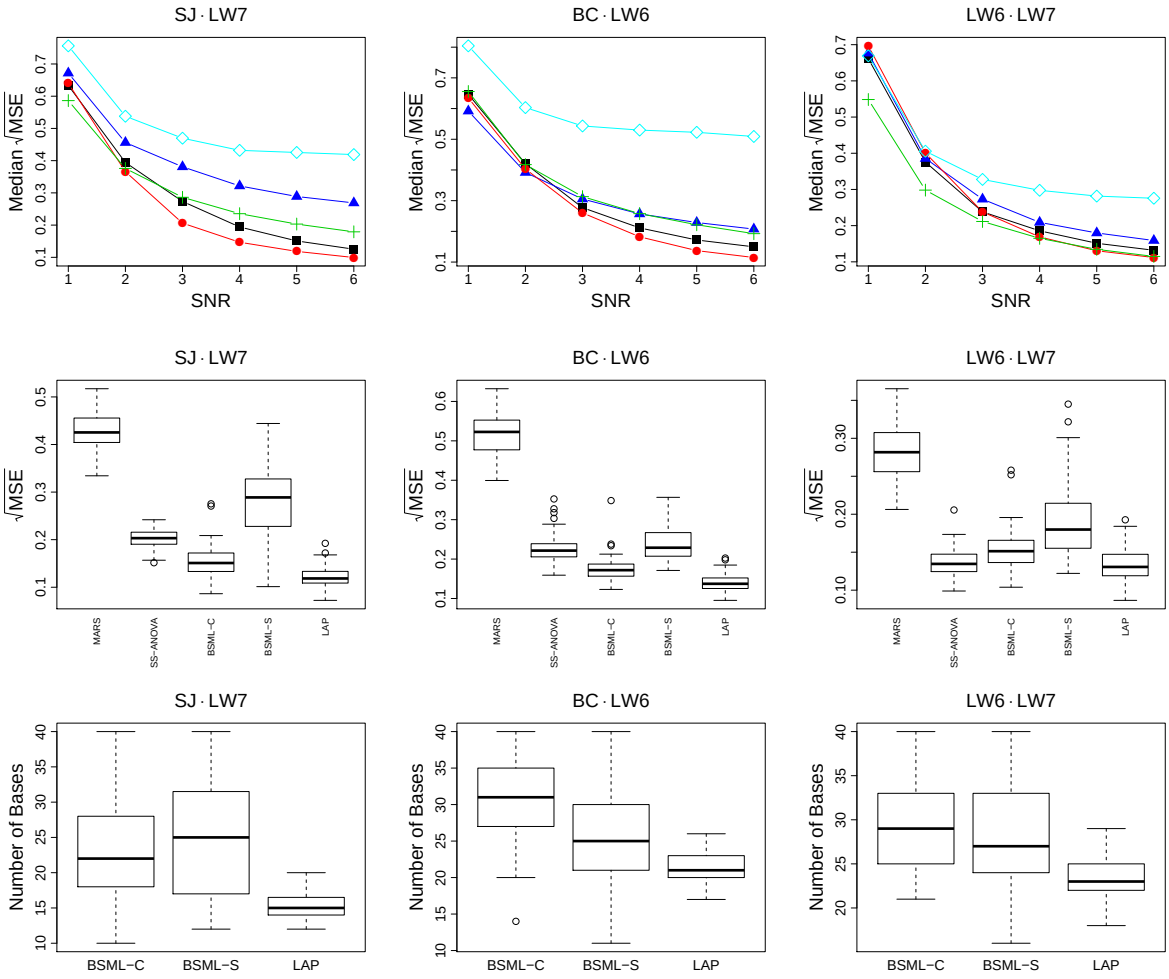


Figure 4.18: Simulation results for the SJ·LW7, BC·LW6, and LW6·LW7 examples with $n = 256$. The top panel shows the median $\sqrt{\text{MSE}}$ across different SNR values for each example, where the symbols used are: ■ for BSML-C, ▲ for BSML-S, ● for LAP, + for SS-ANOVA, and ◇ for MARS. The middle panel shows the boxplots of the $\sqrt{\text{MSE}}$ when SNR $= 5$. The bottom panel shows the number of basis functions selected by BSML-C, BSML-S, and LAP when SNR $= 5$.

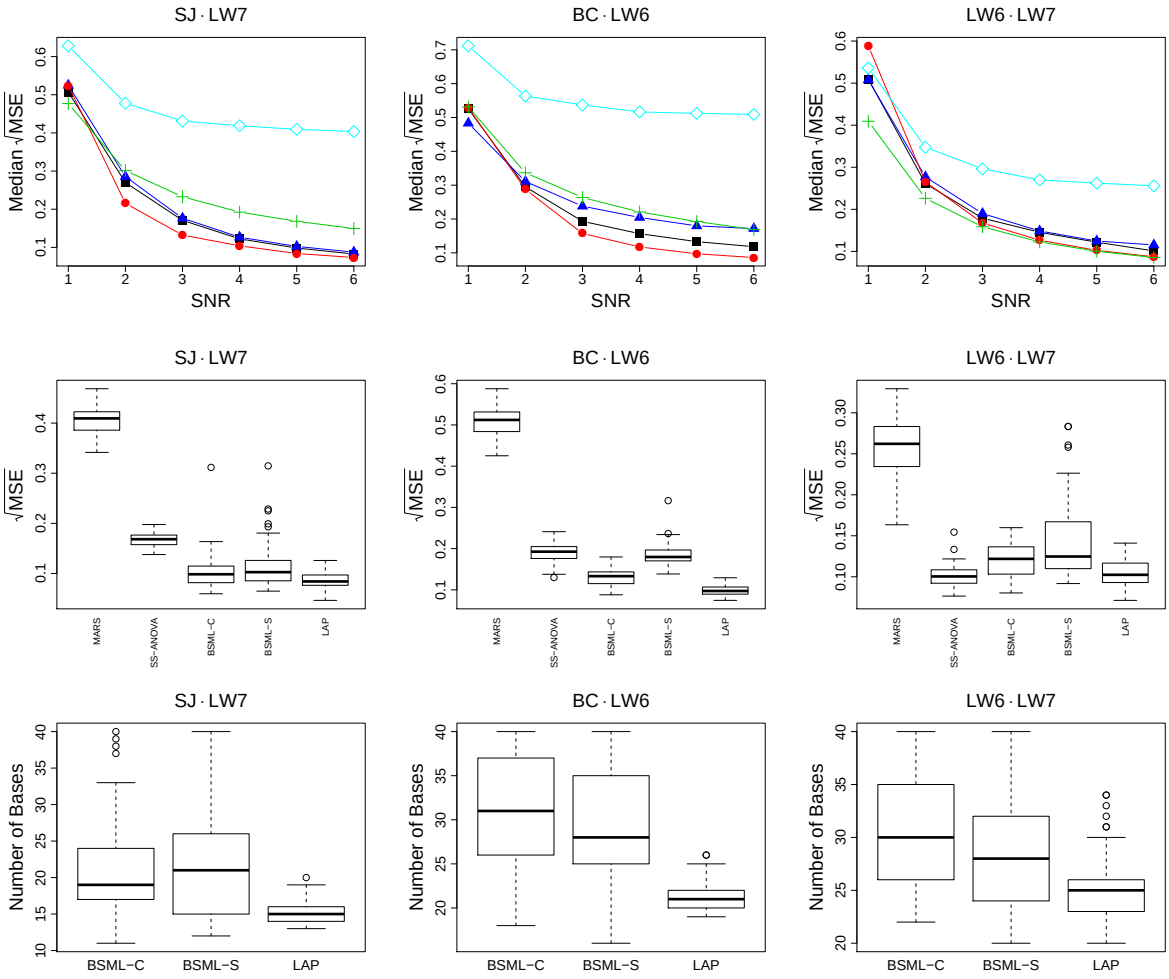


Figure 4.19: Simulation results for the SJ·LW7, BC·LW6, and LW6·LW7 examples with $n = 512$. The top panel shows the median $\sqrt{\text{MSE}}$ across different SNR values for each example, where the symbols used are: ■ for BSML-C, ▲ for BSML-S, ● for LAP, + for SS-ANOVA, and ◇ for MARS. The middle panel shows the boxplots of the $\sqrt{\text{MSE}}$ when SNR $= 5$. The bottom panel shows the number of basis functions selected by BSML-C, BSML-S, and LAP when SNR $= 5$.

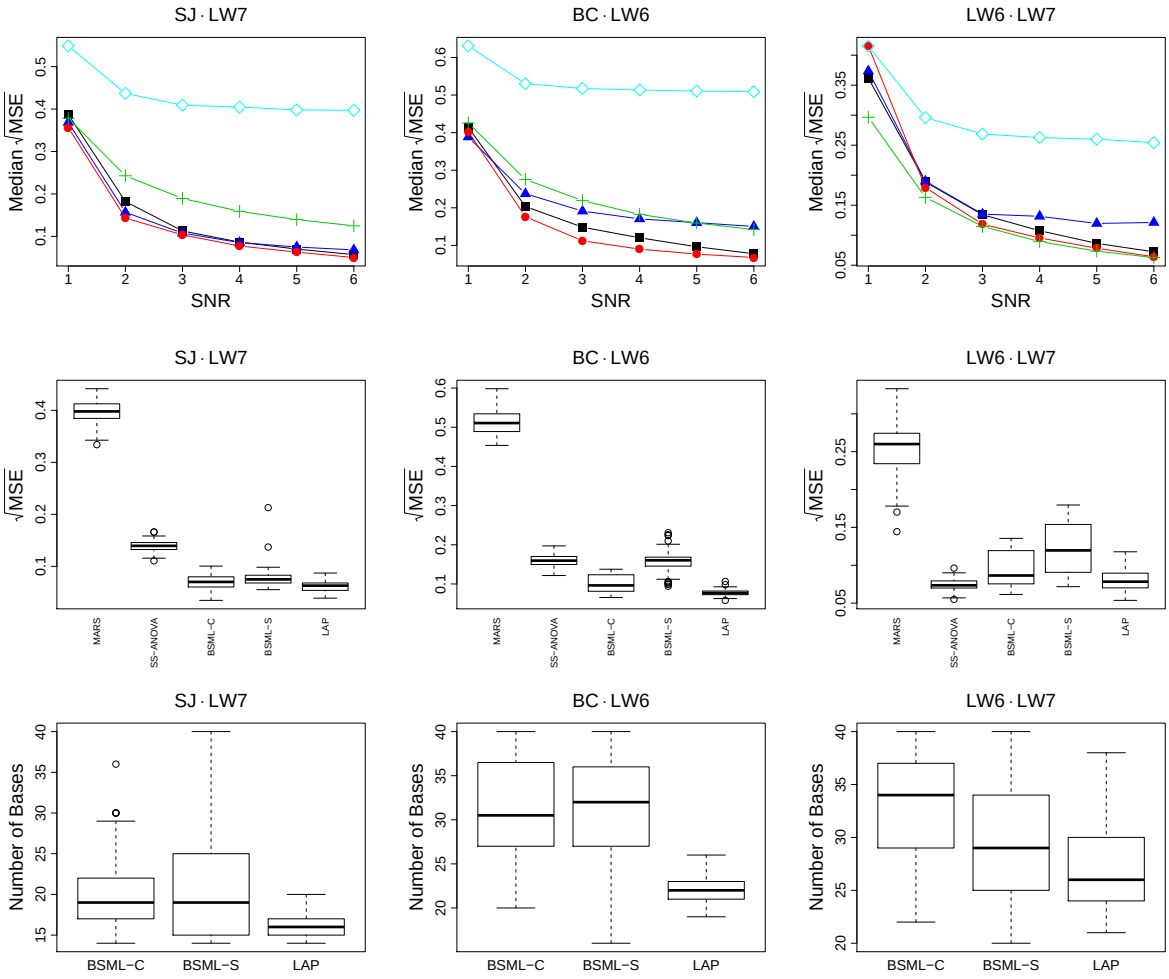


Figure 4.20: Simulation results for the SJ·LW7, BC·LW6, and LW6·LW7 examples with $n = 1024$. The top panel shows the median $\sqrt{\text{MSE}}$ across different SNR values for each example, where the symbols used are: ■ for BSML-C, ▲ for BSML-S, ● for LAP, + for SS-ANOVA, and ◇ for MARS. The middle panel shows the boxplots of the $\sqrt{\text{MSE}}$ when SNR $= 5$. The bottom panel shows the number of basis functions selected by BSML-C, BSML-S, and LAP when SNR $= 5$.

Table 4.11: Average CPU time (in seconds) of fitting the SJ·LW7, BC·LW6, and LW6·LW7 examples using BSML-C, BSML-S, LAP, SS-ANOVA, MARS for sample sizes $n = 256, 512, 1024$ with SNR $= 5$.

	SJ·LW7			BC·LW6		
	$n = 256$	$n = 512$	$n = 1024$	$n = 256$	$n = 512$	$n = 1024$
BSML-C	1.67	3.10	5.18	1.61	3.19	5.20
BSML-S	17.90	34.25	67.16	17.62	34.11	67.55
LAP	3.43	5.00	8.06	4.34	7.73	11.43
SS-ANOVA	1.60	9.65	61.28	1.62	9.59	61.56
MARS	0.02	0.04	0.06	0.02	0.03	0.05

	LW6·LW7		
	$n = 256$	$n = 512$	$n = 1024$
BSML-C	1.32	2.70	4.31
BSML-S	14.74	28.39	56.02
LAP	1.74	3.34	5.46
SS-ANOVA	1.37	7.84	51.29
MARS	0.02	0.03	0.05

4.4 Three Other Examples

Now we consider three other examples where the true function f is not additive in nature. Similar examples can be found in Qiu (2005). Again consider the model in (4.5). The true functions $f(x_1, x_2)$ are listed in Table 4.12 and plotted in Figure 4.21 – 4.23. Table 4.13 – 4.15 contain the basis libraries used in the BSML and LAP procedures for each example.

Table 4.12: Test functions of the Paraboloid-Jump, Mexican hat, and Sinusoid-Jump examples.

Function Name	True Function $f(\boldsymbol{x})$	$SD(f)$
Paraboloid-Jump	$-2\big[(x_1-0.5)^2+(x_2-0.5)^2\big]$ $+0.5\times 1_{[0,0.09)}\big\{(x_1-0.5)^2+(x_2-0.5)^2\big\}$	0.4051
Mexican hat	$(2\pi)^{-1}(2-x_1^2-x_2^2)\exp\big(-0.5(x_1^2+x_2^2)\big)$	0.0498
Sinusoid-Jump	$0.5(1-x_1)x_2+\big[1+0.2\sin(2\pi x_1)\big]1_{[0.6\sin(\pi x_1)+0.2,1]}(x_2)$	0.5716

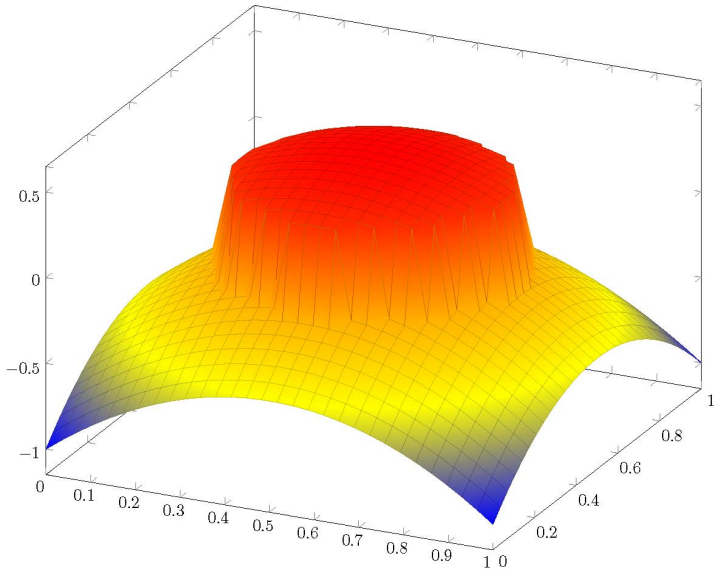


Figure 4.21: Paraboloid-Jump function.

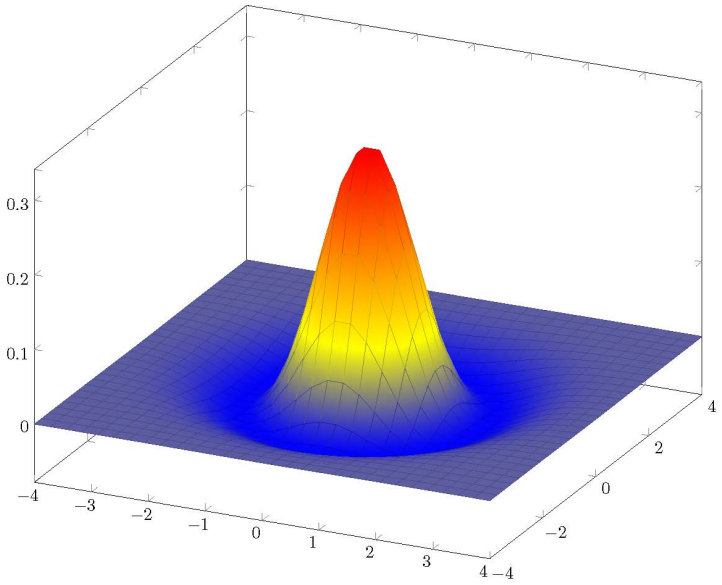


Figure 4.22: Mexican hat function.

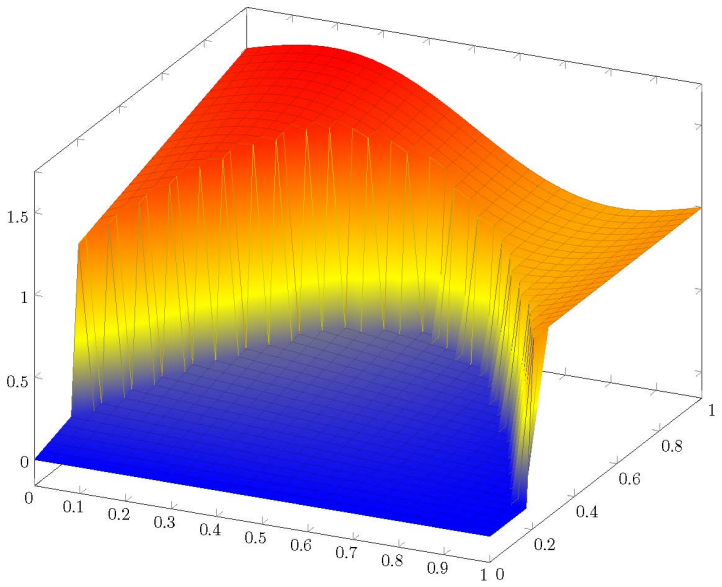


Figure 4.23: Sinusoid-Jump function.

Table 4.13: Basis libraries for the Paraboloid-Jump example.

Basis libraries for the Paraboloid-Jump example
$\mathcal{L}_0 = \{1\}$
$\mathcal{L}_1 = \{x_1, x_1^2, x_1^3\}$
$\mathcal{L}_2 = \{x_2, x_2^2, x_2^3\}$
$\mathcal{L}_3 = \big\{1_{[0,r]}\{(x_1 - c_1)^2 + (x_2 - c_2)^2\}$, where $c_1, c_2 \in \{0.2, 0.3, \ldots, 0.8\}$, $r \in \{0.1, 0.2, \ldots, 0.5\}\big\}$
$\mathcal{L}_4 = \Big\{ \exp\big\{ -0.09\big[(x_1 - c_1)^2 + (x_2 - c_2)^2\big]\big\}$, where $c_1, c_2 \in \{0.2, 0.3, \ldots, 0.8\}\Big\}$
$\mathcal{L}_5 = \Big\{ \exp\big\{ -0.36\big[(x_1 - c_1)^2 + (x_2 - c_2)^2\big]\big\}$, where $c_1, c_2 \in \{0.2, 0.3, \ldots, 0.8\}\Big\}$

Table 4.14: Basis libraries for the Mexican hat example.

Basis libraries for the Mexican hat example
$\mathcal{L}_0 = \{1\}$
$\mathcal{L}_1 = \Big\{ \exp\big\{ -0.09\big[(x_1 - c_1)^2 + (x_2 - c_2)^2\big]\big\}$, where $c_1, c_2 \in \{-3, -2, -1, 0, 1, 2, 3\}\Big\}$
$\mathcal{L}_2 = \Big\{ \exp\big\{ -0.36\big[(x_1 - c_1)^2 + (x_2 - c_2)^2\big]\big\}$, where $c_1, c_2 \in \{-3, -2, -1, 0, 1, 2, 3\}\Big\}$
$\mathcal{L}_3 = \Big\{ \exp\big\{ -\big[(x_1 - c_1)^2 + (x_2 - c_2)^2\big]\big\}$, where $c_1, c_2 \in \{-3, -2, -1, 0, 1, 2, 3\}\Big\}$
$\mathcal{L}_4 = \Big\{ \exp\big\{ -4\big[(x_1 - c_1)^2 + (x_2 - c_2)^2\big]\big\}$, where $c_1, c_2 \in \{-3, -2, -1, 0, 1, 2, 3\}\Big\}$
$\mathcal{L}_5 = \Big\{ \exp\big\{ -9\big[(x_1 - c_1)^2 + (x_2 - c_2)^2\big]\big\}$, where $c_1, c_2 \in \{-3, -2, -1, 0, 1, 2, 3\}\Big\}$

Table 4.15: Basis libraries for the Sinusoid-Jump example.

Basis libraries for the Sinusoid-Jump example
$\mathcal{L}_0 = \{1\}$
$\mathcal{L}_1 = \{x_1, x_2, x_1x_2\}$
$\mathcal{L}_2 = \big\{ \sin(\pi k x_1), \cos(\pi k x_1) : k = 1, 2, 3, 4, 5\big\}$
$\mathcal{L}_3 = 1_{[a\sin(2\pi b^{-1}x_1)+c,1]}(x_2)$where $a, c \in \{0.2, 0.4, 0.6\}, b \in \{1, 2, 4\}$
$\mathcal{L}_4 = \big\{\xi_\ell\xi_\kappa, \text{for } \xi_\ell \in \mathcal{L}_2, \xi_\kappa \in \mathcal{L}_3\big\}$ (tensor product basis functions)
$\mathcal{L}_5 = \big\{\xi_i\xi_j, \text{where } \xi_i \in \{1_{[d_i,d_{i+1}]}(x_1), i = 1, \ldots, 10\}, \xi_j \in \{1_{[d_j,d_{j+1}]}(x_2), j = 1, \ldots, 10\},$ d_i, d_j are the ith and jth elements of the sequence $(0, 0.1, \ldots, 0.9, 1)\big\}$
$\mathcal{L}_6 = \mathcal{C}_2(x_1)$, with knots $\{0.02 + 0.01j, j = 1, \ldots, 95\}$

For the Paraboloid-Jump example, $\mathcal{L}_3$ contains the 2d step functions which have jumps for points inside a circle with different center locations and radii. Bases from $\mathcal{L}_3$ are used to capture the jump in the Paraboloid-Jump function. $\mathcal{L}_4$ and $\mathcal{L}_5$ contains radial basis functions which have shapes close to the paraboloid.

For the Mexican hat example, the nonnull libraries contains radial basis functions with different centers and shape parameters. These bases can be used to approximate the Mexican hat function well because both are closely related to the Gaussian density.

For the Sinusoid-Jump example, $\mathcal{L}_3$ contains the 2d step functions which have jumps for x_2 is greater or equal to the sine functions of x_1 with different amplitudes, periods, and shifts. These basis functions are used to capture the jump in the Sinusoid-Jump example. We also included another type of step functions, which are the 2d B-spline function with degree equal to zero contained in $\mathcal{L}_5$. If a basis selection procedure works well, then it should not select a basis from $\mathcal{L}_5$ since none of the bases from $\mathcal{L}_5$ is used to construct the Sinusoid-Jump function. To fit the continuous part of the Sinusoid-Jump

function, we have the Fourier series in $\mathcal{L}_2$, the tensor product basis functions in $\mathcal{L}_4$, and the cubic spline representers of x_1 in $\mathcal{L}_6$. A good basis selection procedure should select a single basis function from $\mathcal{L}_4$, and none from $\mathcal{L}_6$.

In the simulation, the design points (x_{i1}, x_{i2}) for $i = 1, \ldots, n$ are uniformly sampled from $[0,1]\times[0,1]$ for the Paraboloid-Jump and Sinusoid-Jump examples. For the Mexican hat example, they are uniformly sampled from $[-4,4] \times [-4,4]$. The sample size used is $n \in \{256, 512, 1024\}$. The error variance σ^2 is chosen such that SNR $\in \{1,2,3,4,5,6\}$. For the BSML and LAP procedures, the maximal number of bases M is fixed at 30. For the look-ahead procedure, the IDF spaces are $S_i = \{1,2,3\}$ for each library $\mathcal{L}_i$. M_d is set to be $\prod_{i=1}^{5} |S_i| = 3^5 = 243$ for the Paraboloid-Jump and Mexican hat examples, and $\prod_{i=1}^{6} |S_i| = 3^6 = 729$ for the Sinusoid-Jump example. The 10-fold cross validation is used to select the final model in the look-ahead procedure.

We also applied the SS-ANOVA model and the MARS procedure to these three examples. For all examples, SS-ANOVA model assumes $f \in W_2^2[0,1] \otimes W_2^2[0,1]$, where the support for the Mexican hat example has been scaled to $[0,1]\times[0,1]$. For the MARS procedure, we set `degree = 2` in `mars` which means we allow pairwise interactions terms in the model. The IDF used is fixed at 3, and the maximum number of model terms `nk` is fixed at 30. The simulation results for BSML-C, BSML-S, LAP, SS-ANOVA, and MARS based on 100 random samples are shown in Figure 4.24 – 4.26. The average CPU time for each sample size is listed in Table 4.16.

Based on the results, MARS did not work well for all three examples and SS-ANOVA works relatively well only for the Mexican hat example. LAP has the best performance for the Sinusoid-Jump example when SNR ≥ 3 across all sample sizes. For the other two examples, LAP has similar performance as the BSML procedures. In terms of computation speed, LAP is faster than BSML-S for all examples, and is similar to BSML-C except for the Sinusoid-Jump example.

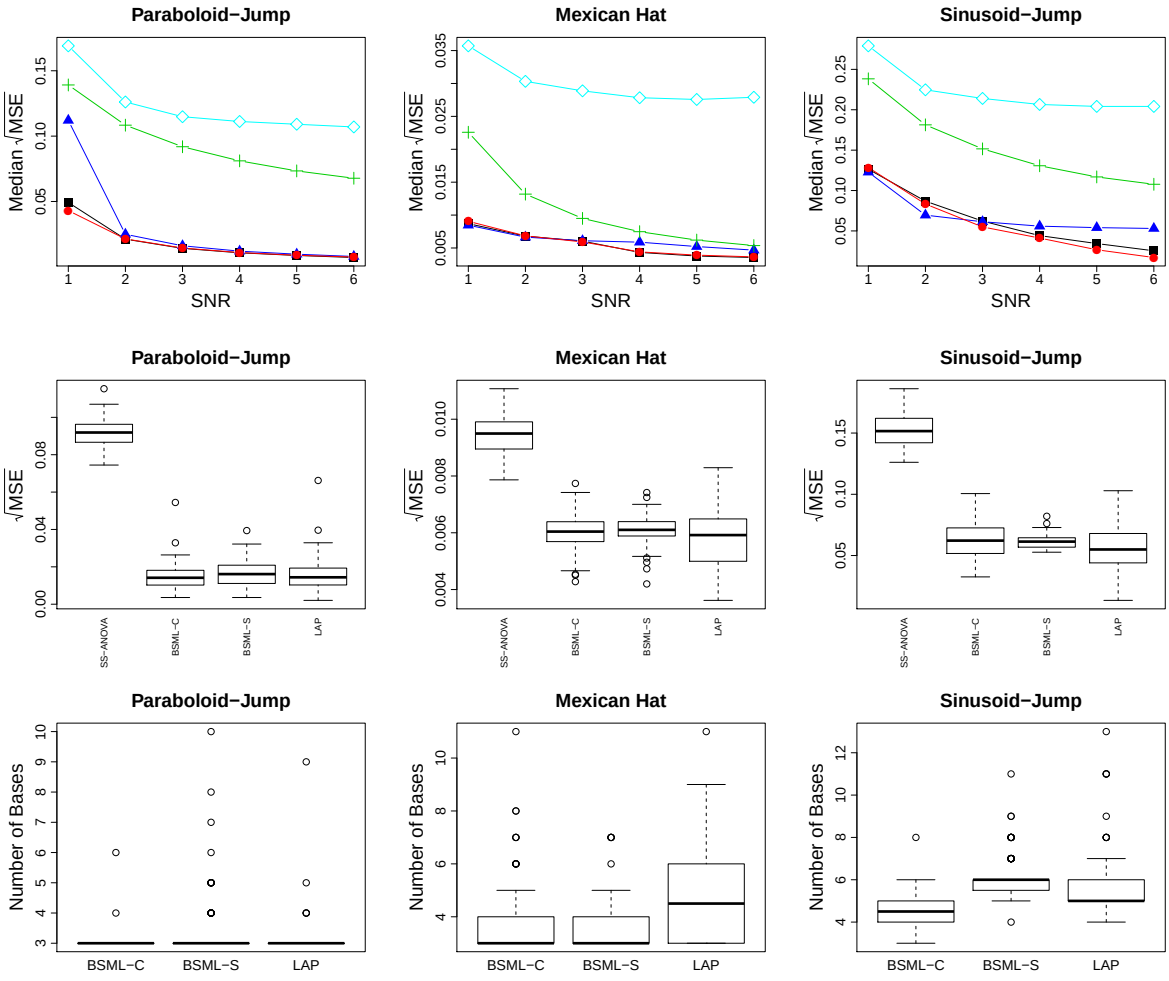


Figure 4.24: Simulation results for the Paraboloid-Jump, Mexican hat, and Sinusoid-Jump examples with $n = 256$. The top panel shows the median $\sqrt{\text{MSE}}$ across different SNR values for each example, where the symbols used are: ■ for BSML-C, ▲ for BSML-S, ● for LAP, + for SS-ANOVA, and ◇ for MARS. The middle panel shows the boxplots of the $\sqrt{\text{MSE}}$ when SNR $= 3$. The results of fitting all three examples with MARS are not shown here because they are much worse compared to the results from the BSML and LAP procedures. The bottom panel shows the number of basis functions selected by BSML-C, BSML-S, and LAP when SNR $= 3$.

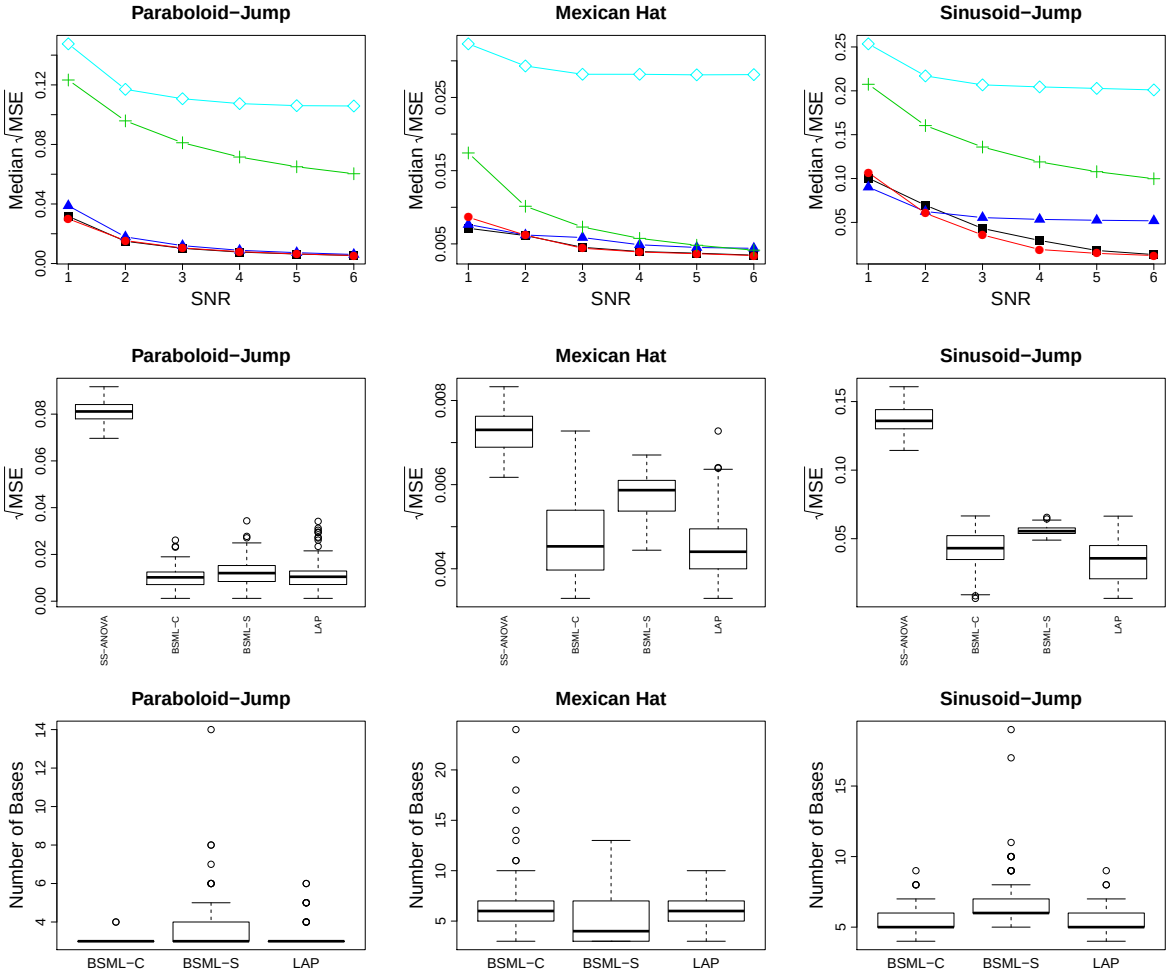


Figure 4.25: Simulation results for the Paraboloid-Jump, Mexican hat, and Sinusoid-Jump examples with $n = 512$. The top panel shows the median $\sqrt{\text{MSE}}$ across different SNR values for each example, where the symbols used are: ■ for BSML-C, ▲ for BSML-S, ● for LAP, + for SS-ANOVA, and ◇ for MARS. The middle panel shows the boxplots of the $\sqrt{\text{MSE}}$ when SNR $= 3$. The results of fitting all three examples with MARS are not shown here because they are much worse compared to the results from the BSML and LAP procedures. The bottom panel shows the number of basis functions selected by BSML-C, BSML-S, and LAP when SNR $= 3$.

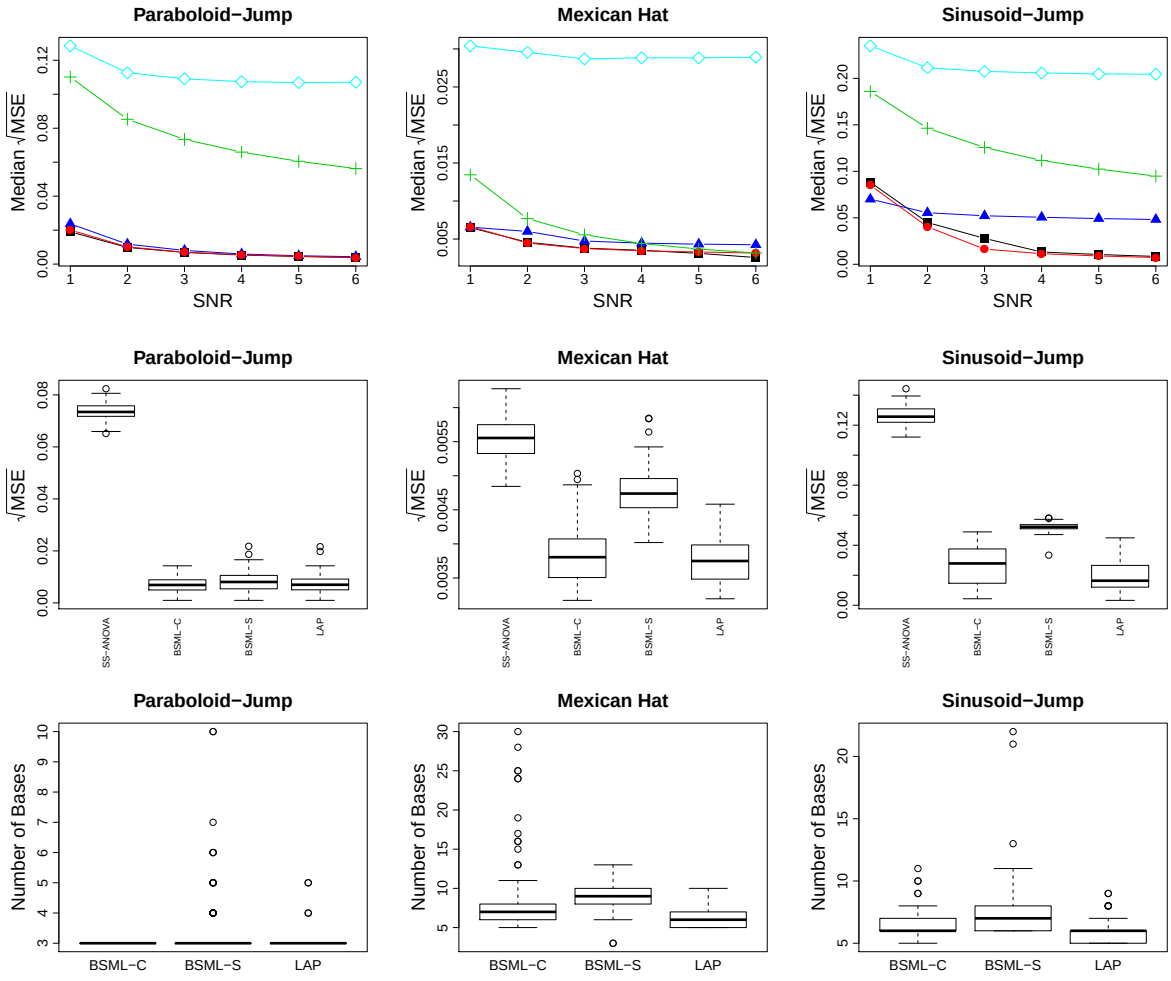


Figure 4.26: Simulation results for the Paraboloid-Jump, Mexican hat, and Sinusoid-Jump examples with $n = 1024$. The top panel shows the median $\sqrt{\text{MSE}}$ across different SNR values for each example, where the symbols used are: ■ for BSML-C, ▲ for BSML-S, ● for LAP, + for SS-ANOVA, and ◇ for MARS. The middle panel shows the boxplots of the $\sqrt{\text{MSE}}$ when SNR $= 3$. The results of fitting all three examples with MARS are not shown here because they are much worse compared to the results from the BSML and LAP procedures. The bottom panel shows the number of basis functions selected by BSML-C, BSML-S, and LAP when SNR $= 3$.

Table 4.16: Average CPU time (in seconds) of fitting the Paraboloid-Jump, Mexican hat, and Sinusoid-Jump examples using BSML-C, BSML-S, LAP, SS-ANOVA, MARS for sample sizes $n = 256, 512, 1024$ with SNR $= 3$.

	Paraboloid-Jump			Mexican hat		
	$n = 256$	$n = 512$	$n = 1024$	$n = 256$	$n = 512$	$n = 1024$
BSML-C	1.63	3.06	6.57	1.08	2.12	4.27
BSML-S	15.21	31.62	52.54	13.39	28.64	43.03
LAP	1.40	2.47	4.71	1.55	2.39	3.41
SS-ANOVA	0.85	4.56	36.31	0.47	2.08	9.16
MARS	0.01	0.02	0.03	0.02	0.02	0.04

	Sinusoid-Jump		
	$n = 256$	$n = 512$	$n = 1024$
BSML-C	2.37	4.48	9.71
BSML-S	21.64	43.28	74.05
LAP	6.41	10.13	16.31
SS-ANOVA	0.97	5.50	50.76
MARS	0.01	0.02	0.03

To sum up, based on the simulations presented in Sections 4.1 – 4.4, we conclude that our look-ahead procedure has similar or better performance compared to the BSML procedures, and it tends to select fewer basis functions on average. Moreover, in terms of the computation speed our procedure is superior than BSML-S, and has the edge over BSML-C in some situations.

Chapter 5

Applications

5.1 Well Log

The well log data were introduced in Ruanaidh and Fitzgerald (1996). These data contain measurements of magnetic resonance of the rock strata at different time points while drilling a well. The response magnetic resonance is in the scale of 10^4. The data set is available at `http://mldata.org/repository/data/viewslug/well-log/`. Because the data contain a considerable number of outliers, we removed the 40 smallest observations as done similarly in Ruanaidh and Fitzgerald (1996). The adjusted data set contains 4010 observations and is plotted in Figure 5.1. The explanatory variable is scaled into $x = [0, 1]$. Here is the `R` code for loading the data set and creating the variables.

```
library(bsml)
library(assist)
source("LAP.R")

wl <- read.csv("well-log.csv", header=FALSE)
N <- nrow(wl)
outliers <- order(wl[,1])[1:40]
y <- wl[-outliers,1]/(10^4)
```

```
n <- length(y)
time <- c(1:N)[-outliers]
x <- (time-min(time))/(max(time)-min(time))

plot(time, y, type="l", ylab="Magnetic Resonance", xlab="Time Points")
```

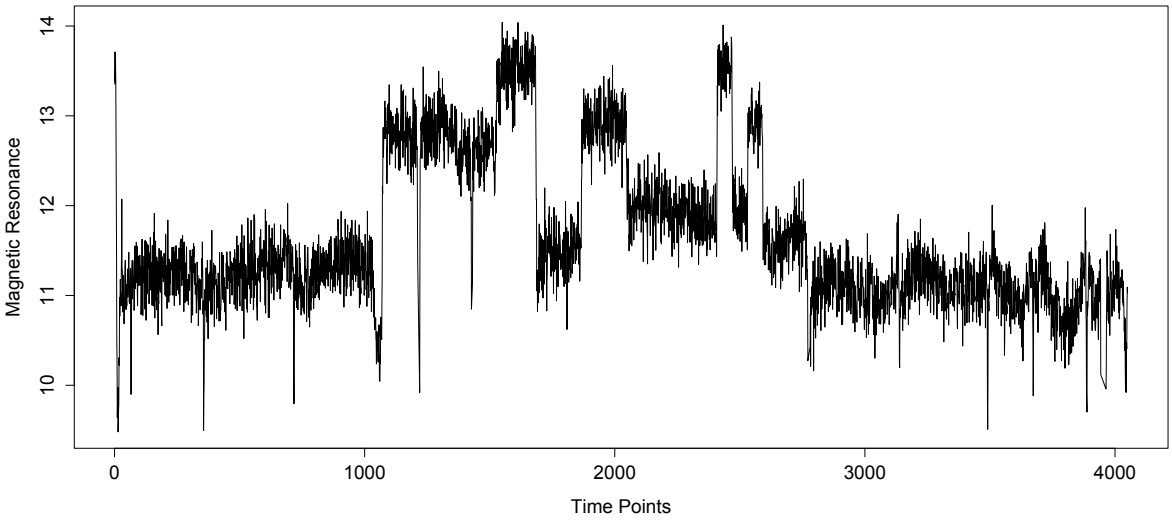


Figure 5.1: Well log data

We first fit a cubic spline to the data using the `R` function `ssr` in the `assist` package, where the smoothing parameter is selected by the generalized maximum likelihood (GML) criterion. The fit is shown in Figure 5.2. Here is the R code we used:

```
fit.ssr <- ssr(y~x, cubic(x), spar="m")

plot(time, y, col="gray", ylab="Magnetic Resonance", xlab="Time Points",
     main="Cubic Spline", xaxt="n", cex.lab=1.5, cex.main=2)
axis(1, at=seq(0,4100, by=200), label=seq(0,4100, by=200))
lines(time, fit.ssr$fit, lwd=3, col=3)
```

We can see that cubic spline overfits the data. For example, at time points around 1210, and 1430 the fit is clearly affected by some outliers.

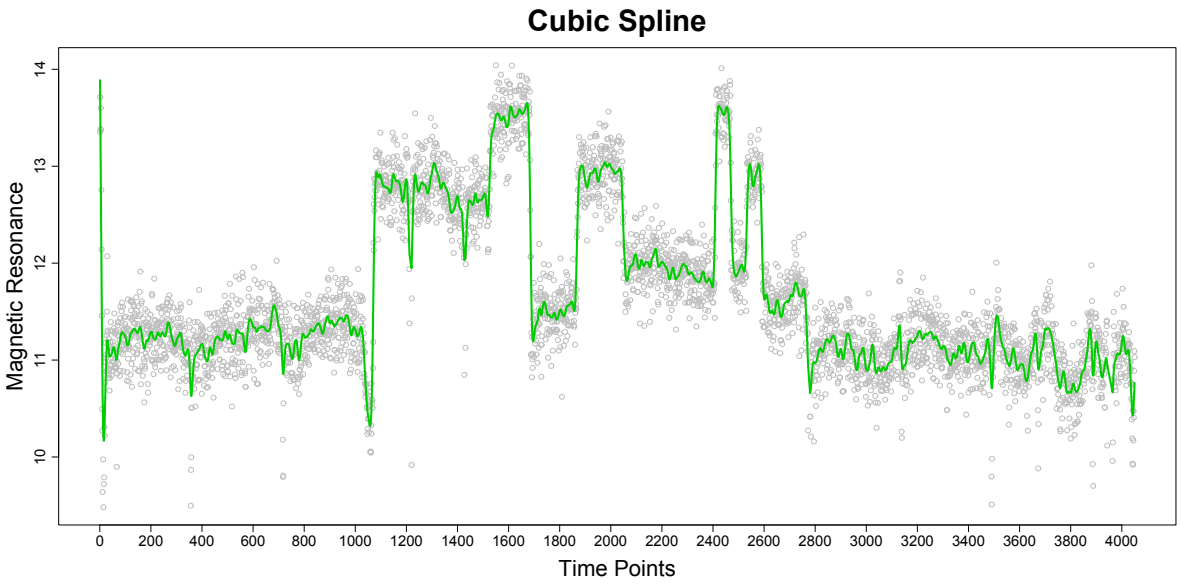


Figure 5.2: Cubic spline fits of the well log data. The Gray dots are the observations. The green line is the cubic spline fit with the smoothing parameter selected by GML criterion.

Next we fit the data using BSML-C, BSML-S and LAP. Table 5.1 lists the libraries of basis functions used in these procedures.

Table 5.1: Basis libraries for the well log example.

$\mathcal{L}_0 = \{1, x\}$
$\mathcal{L}_1 = \mathcal{C}_2(x)$, with knots $\{0.01j, j = 1, \ldots, 99\}$
$\mathcal{L}_2 = \mathcal{T}_0(x) = \left\{1_{(z_1,1]}(x), \ldots, 1_{(z_{99},1]}(x)\right\},\ z_j = 0.01j,\ j = 1, \ldots, 99$
$\mathcal{L}_3 = \mathcal{T}_1(x) = \left\{(x - z_1)_+, \ldots, (x - z_{99})_+\right\},\ z_j = 0.01j,\ j = 1, \ldots, 99$
$\mathcal{L}_4 = \mathcal{T}_2(x) = \left\{(x - z_1)_+^2, \ldots, (x - z_{99})_+^2\right\},\ z_j = 0.01j,\ j = 1, \ldots, 99$

Here is how we generate the libraries of basis functions in R:

```
loc <- seq(0.01,0.99,0.01)
```

```
baseslist.gen <- function(x, loc){
 n <- length(x)
 pt <- rep(1,n)%o%loc
 L0 <- cbind(1, x)
 L1 <- cubic(x,t=loc)
 L2 <- 1*(x>pt)
 L3 <- (x-pt)*(x>pt)
 L4 <- (x-pt)^2*(x>pt)
 baseslist <- list(L0, L1, L2, L3, L4)
 return(baseslist)
}

baseslist <- baseslist.gen(x, loc)
```

For the BSML and LAP procedures, we specified $M = 100$, which allows at most 100 basis functions to be selected in the final model (including the null bases). Here are the R codes for fitting the data using BSML-C and BSML-S.

```
# BSML-C:
fit.bsmlc <- bsml(y, baseslist, method="bsmlc", maxbas=100)

# BSML-S:
fit.bsmls <- bsml(y, baseslist, method="bsmls", maxbas=100)

# Figure 5.3:
par(mfrow=c(2,1))
plot(time, y, col="gray", ylab="Magnetic Resonance", xlab="Time Points",
     main="BSML-C", xaxt="n", cex.lab=1.5, cex.main=2)
axis(1, at=seq(0,4100, by=200), label=seq(0,4100, by=200))
lines(time, fit.bsmlc$fit, lwd=3, col=1)

plot(time, y, col="gray", ylab="Magnetic Resonance", xlab="Time Points",
     main="BSML-S", xaxt="n", cex.lab=1.5, cex.main=2)
axis(1, at=seq(0,4100, by=200), label=seq(0,4100, by=200))
lines(time, fit.bsmls$fit, lwd=3, col=4)
```

The BSML-C and BSML-S fits are shown in Figure 5.3. The BSML-C procedure also seem to overfit the data. The fit contains many small jumps besides the obvious big jumps. In comparison, the BSML-S procedure gives a smoother fit. BSML-C has

selected 90 basis functions, among which 55 of them are from $\mathcal{L}_2$. BSML-S has selected 90 basis functions, among which 20 of them are from $\mathcal{L}_2$.

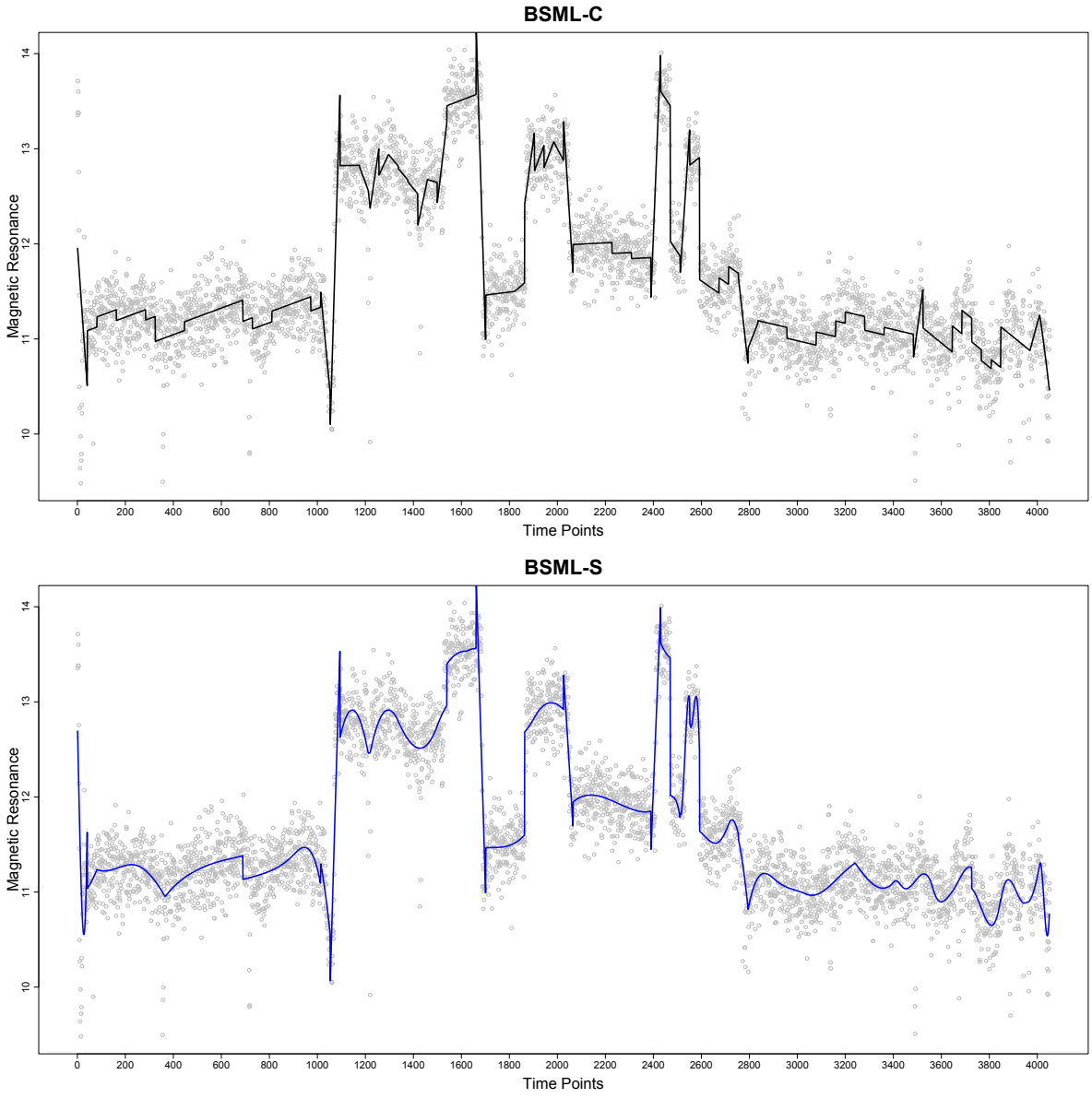


Figure 5.3: BSML-C and BSML-S fits of the well log data. The Gray dots are the observations. The black line in the top panel is the BSML-C fit. The blue line in the lower panel is the BSML-S fit.

For the LAP, the IDF values used are $\{1, 2, 3, 4, 5, 6\}$ for each library, since this works well for the Sine-Jumps and Blocks-Curves examples in our simulation. The input IDF matrix contains in total $6^4 = 1296$ IDF tuning vectors. By the default setting in LAP, $M_d = 1296$, so that given any tuning vector in the IDF matrix, there exists a unique candidate model generated from the forward selection process. Figure 5.4 shows the LAP fit together with the 95% bootstrap confidence interval. Here for the confidence interval we specified the number of bootstrap replicates to be `bootrep` $= 200$. The R code used is shown below.

```
# LAP:
nlib <- length(baseslist)
idfs <- append(list(1),rep(list(c(1:6)),nlib-1))
idfs <- as.matrix(expand.grid(idfs))

set.seed(123)
fit.lap <- LAP(y, baseslist, maxbas=100, idfs=idfs, criterion="BIC")

# Confidence interval for LAP:
pred.lap <- predict.LAP(fit.lap, bases.include=c(1,1,1,1,1,1),
                        new.baseslist=baseslist, confint=T)

# Figure 5.4:
par(mfrow=c(1,1))
plot(time, y, col="gray", ylab="Magnetic Resonance", xlab="Time Points",
     main="LAP", xaxt="n", cex.lab=1.5, cex.main=2)
axis(1, at=seq(0,4100, by=200), label=seq(0,4100, by=200))
lines(time, pred.lap$fit, lwd=3, col=2)
lines(time, pred.lap$lower, col=1,  lwd=2, lty=2)
lines(time, pred.lap$upper, col=1,  lwd=2, lty=2)
```

The LAP fit has the most parsimonious representations in terms of the basis functions since only 41 bases are selected, among which 13 of them are from $\mathcal{L}_2$. LAP has successfully identified the major jumps and does not overfit the data by adding small jumps like BSML-C does. Although BSML-S does a better job than BSML-C, its fit is still affected by some outliers when compared to LAP, for example at time points around

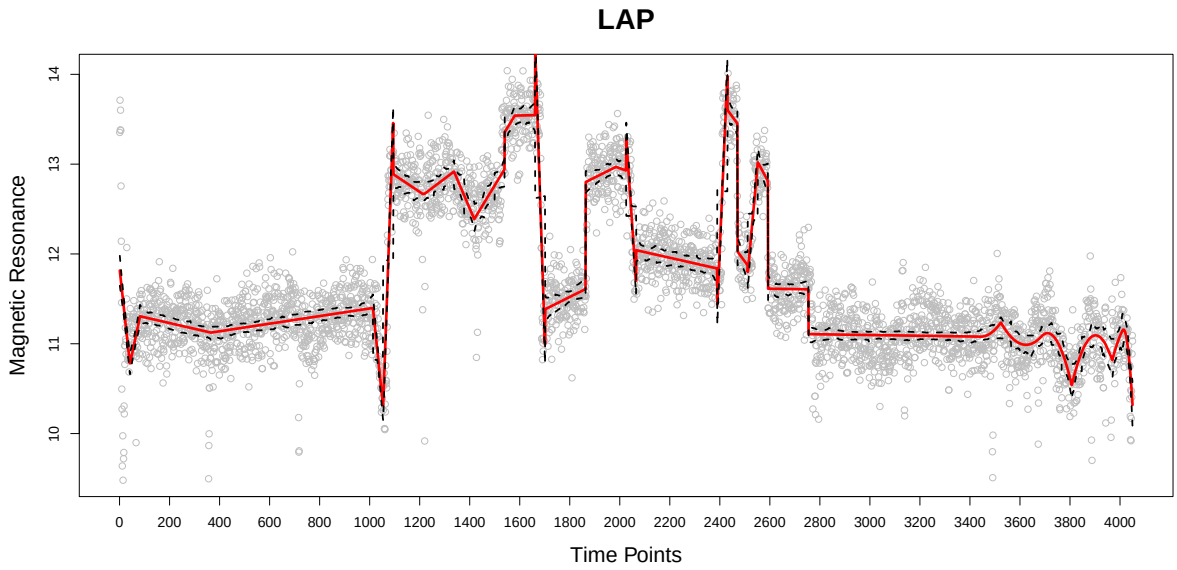


Figure 5.4: LAP fit of the well log data. The Gray dots are the observations. The red solid line is the LAP fit. The black dashed lines are the lower and upper bounds of the 95% bootstrap confidence interval.

40 and 1010.

5.2 Ozone Pressure

The Antarctic ozone hole refers to the ozone depletion observed in the stratosphere of the antarctic region. This phenomenon has be widely studied, for example see Solomon (1999). Ozone depletion in Antarctica was first documented in 1985, and is observed during the Spring season. We will use the ozonesonde data collected at the Amundsen-Scott South Pole Station to investigate this phenomenon. The data is available on the National Oceanic and Atmospheric Administration website: `ftp://ftp.cmdl.noaa.gov/ozwv/Ozonesonde/SouthPole,Antartica/100MeterAverageFiles/`. The raw data between September 2006 and August 2009 contains 0.1-km vertical averages of the ozone partial pressure (in mPa) for 207 unique ozonesonde launch dates. We will focus on the

altitude levels (in km) within the range 11 km to 30 km, where the lower stratosphere is located. Among the total 207 launch dates, we selected 181 dates for which few than 20 values are missing for the ozone partial pressure at the altitude levels of interest.

For our analysis, the response variable is the ozone partial pressure. The covariates are the altitude (from 11 km to 30 km for every 0.1 km, in total 191 levels) and the dates of the records. For convenience, we treat the dates as numerical by counting the number of days elapsed since the start of the records, which is September 06, 2006. We call this variable "day". For example, for the first record, day = 1. For the second record that occurred on September 09, 2006, day = 4, and so on. The variable "day" has 181 unique values.

Instead of including all the 191 different altitude levels in our analysis, we used two subsets of them. The first subset consists of altitude levels $\{10 + j : j = 1, \ldots, 20\}$. The second subset consists of altitude levels $\{10.5 + j : j = 1, \ldots, 19\}$. All the observations with altitude levels in the first subset forms the training set. All the observations with altitude levels in the second subset forms the test set. We will build models based on the training set, and test their performances using the test set. For the training set, 91 out of 3620 observations are missing. For the test set, 54 out of 3439 observations are missing. Most of the missing values occur at upper altitudes on a subset of dates. We scale the covariates `altitude` and `day` in both the training set and the test set to $[0, 1]$. Denote x_1 as the scaled `altitude` and x_2 as the scaled `day`.

In the `R` code below, `alt.train` and `alt.test` are the first and the second subsets of distinct altitude levels. `date.uni` contains unique values of the variable `day`. `ozone.train` is a 20×181 matrix that contains observed ozone partial pressure from the training set. `ozone.test` is a 19×181 matrix that contains observed ozone partial pressure from the test set. The (i, j)th element in `ozone.train` and `ozone.test` corresponds to the ozone pressure observed at the ith altitude on jth ozonesonde launch date. The

missing values in `ozone.train` are removed to form the 3529×1 vector `ozone`. `index.na` contains the indices of these missing values. `x1` and `x2` are 3529×1 vectors that contain the scaled `altitude` and the scaled `day` respectively for each observation in `ozone`. `x1.test` and `x2.test` are the scaled covariates for the observations in `ozone.test`. The code used to generate `x1`, `x2`, `x1.test`, and `x2.test` is given below.

```
library(assist)
library(bsml)
library(gss)
library(lattice)
source("LAP.R")

alt.train <- seq(from=11, to=30, by=1)
n.date <- length(date.uni)
n.alt <- length(alt.train)

index.na <- which(is.na(ozone.train))
ozone <- na.omit(as.vector(ozone.train))
n <- length(ozone)

covar <- as.matrix(expand.grid(alt.train, date.uni))
var1 <- covar[-index.na,1]
var2 <- covar[-index.na,2]

x1 <- (var1-min(var1))/(max(var1)-min(var1))
x2 <- (var2-min(var2))/(max(var2)-min(var2))

alt.test <- seq(from=11.5, to=29.5, by=1)
covar.test <- expand.grid(alt.test, date.uni)
x1.test <- (covar.test[,1]-min(var1))/(max(var1)-min(var1))
x2.test <- (covar.test[,2]-min(var2))/(max(var2)-min(var2))
```

The following `R` code is used to generate the level plot of the ozone partial pressure, shown in Figure 5.5. Here `month.tick` and `month.labels` are the tick mark locations and labels for the x-axis of the level plot. The level plot illustrates that the lowest ozone partial pressure occurred between 14 km and 20 km in September, October, and November. In addition, the frequency of launches shows seasonality. Among all the launches of interest,

the months with most frequent launches were September and October, with a total of 30 and 31 launches respectively. In contrast, the month with fewest launches was April, which had only 6 launches in total.

```
# Figure 5.5:
levelplot(t(ozone.train), aspect=0.5,
  ylab=list(label="Altitude (km)", cex=1.5),
  xlab=list(label="Month/Year", cex=1.5),
  main=list(label="Ozone Pressure from Sep 2006
                  to Aug 2009 \n (training set)", cex=2),
  col.regions = topo.colors(300),
  at=seq(min(ozone.train, na.rm=TRUE),
         max(ozone.train, na.rm=TRUE), length=300),
  colorkey=list(tick.number=10, labels=list(cex=1.2)),
  scales=list(x=list(cex=1, at=month.tick, labels=month.labels),
              y=list(cex=1.4), at=1:n.alt))
```

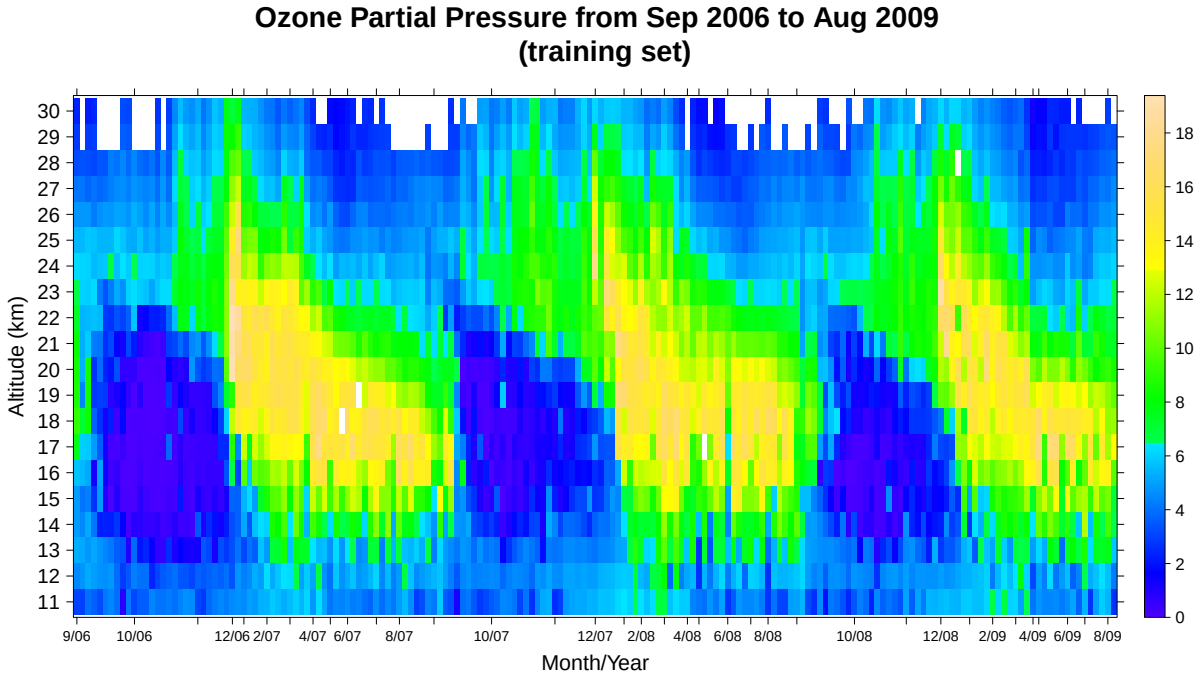


Figure 5.5: Level plot of the ozone partial pressure at altitudes 11 km, 12 km, ..., 30 km, for 181 launch dates between September 2006 and August 2009.

We first fit an SS-ANOVA model with the model space $W_2^2[0,1] \times W_2^2(per)$ to the training set, using the `R` package `gss` (Gu (2013a)). The smoothing parameter is selected by the generalized maximum likelihood (GML) criterion. The following `R` code is used:

```
set.seed(321)
# SS-ANOVA:
z2 <- 3*x2
fit.gss <- ssanova(ozone~x1*z2, type=list(x1=list("cubic", c(0,1)),
                   z2=list("per", c(0,3))), nbasis=100, method="m")
```

Here we multiply x_2 by 3 to make the period of the periodic spline representers to be $1/3$, so the data for each year are fitted using the same periodic spline representers. In order to make fair comparisons between the performance of the SS-ANOVA model and that of the BSML and LAP procedures, we specified `nbasis=100`, so 100 randomly selected knots are used in the SS-ANOVA model.

Next we fit the training data using the BSML and LAP procedures. Table 5.2 lists the libraries of basis functions used in these procedures. Here $\mathcal{L}_1$ contains cubic spline representers for the altitude, where the knots used are the unique values of x_1. $\mathcal{L}_2$ contains Fourier series with the frequencies equal to multiples of 3, because the data are from three years. The bases in $\mathcal{L}_2$ can be used to model the seasonal main effect of `day` on the ozone partial pressure. $\mathcal{L}_3$ contains the basis functions that are tensor products of the bases from $\{x_1 - 0.5\} \cup \mathcal{L}_1$ and the bases from $\mathcal{L}_2$. The bases in $\mathcal{L}_3$ can be used to model the interaction effect of the covariates `altitude` and `day` on the ozone partial pressure. There are 462 basis functions in total.

Table 5.2: Basis libraries for the ozone example.

$\mathcal{L}_0 = \{1, x_1 - 0.5\}$
$\mathcal{L}_1 = \mathcal{C}_2(x_1)$, with knots $\big\{(j-1)/19, j = 1, \ldots, 20\big\}$
$\mathcal{L}_2 = \big\{\sin(6\pi k x_2), \cos(6\pi k x_2) : k = 1, 2, \ldots, 10\big\}$
$\mathcal{L}_3 = \big\{\xi_i \xi_j : \xi_i \in \{x_1 - 0.5\} \cup \mathcal{L}_1,\ \xi_j \in \mathcal{L}_2\big\}$

The following code is used to generate the basis libraries for the training set and the test set:

```
baseslist.gen <- function(x1, x2, loc){
  # L0:
  L.null <- cbind(1, x1-0.5)
  # L1:
  L.cubic <- cubic(x1, t=loc)
  # L2:
  kappa <- c(1:10)
  L.fou <- cbind(sin(6*pi*x2%o%kappa),cos(6*pi*x2%o%kappa))
  # L3:
  L.prod <- (cbind(x1-0.5,L.cubic)%x%matrix(1,ncol=ncol(L.fou)))*
              (matrix(1,ncol=ncol(cbind(x1-0.5,L.cubic)))%x%L.fou)
  baseslist <- list(L.null, L.cubic, L.fou, L.prod)
  return(baseslist)
}

# libraries of basis functions for the training set:
baseslist <- baseslist.gen(x1, x2, loc=unique(x1))

# libraries of basis functions for the test set:
baseslist.test <- baseslist.gen(x1.test, x2.test, loc=unique(x1))
```

For the BSML and LAP procedures, we specified the maximum number of bases that can be selected to be $M = 100$. The following code is used to fit the training data with the BSML procedures.

```
set.seed(321)
# BSML-C:
```

```
fit.bsmlc <- bsml(ozone, baseslist, method="bsmlc", maxbas=100)

# BSML-S:
fit.bsmls <- bsml(ozone, baseslist, method="bsmls", maxbas=100)
```

For LAP, we first tried the IDF values $\{0.5, 1, 1.5, 2\}$ for each library, so the input IDF matrix contains in total $4^3 = 64$ IDF tuning vectors. By the default setting in LAP, $M_d = 64$, so that each tuning vector in the IDF matrix corresponds to a unique candidate model generated from the forward selection process. The fitted object is named `fit1.lap` in the code below. The IDF values selected by cross validation in `fit1.lap` are 1, 0.5, and 0.5 for $\mathcal{L}_1$, $\mathcal{L}_2$, and $\mathcal{L}_3$ respectively. Therefore, we adjusted the IDF values to be $\{0.2, 0.5, 0.8, 1\}$ for each library, and refit the training data with LAP to obtain the final model. The second fitted objected is named `fit2.lap`. Given below is the code for fitting the training data twice with LAP.

```
idfs1 <- append(list(1),rep(list(c(0.5, 1, 1.5, 2)),nlib-1))
idfs1 <- as.matrix(expand.grid(idfs1))

set.seed(321)
fit1.lap <- LAP(ozone, baseslist, maxbas=100,
                idfs=idfs1, criterion="BIC")
fit1.lap$idfs.cv

## Adjust the IDF tuning values and refit with LAP:

idfs2 <- append(list(1),rep(list(c(0.2, 0.5, 0.8, 1)),nlib-1))
idfs2 <- as.matrix(expand.grid(idfs2))

set.seed(321)
fit2.lap <- LAP(ozone, baseslist, maxbas=100,
                idfs=idfs2, criterion="BIC")
```

Figures 5.6, 5.7, 5.8, and 5.9 are the level plots of the fitted ozone partial pressure at 191 equally spaced grid points for `altitude` and 218 equally spaced grid points for `day`, by using SS-ANOVA model, BSML-C, BSML-S, and LAP procedures respectively.

Figures 5.10 and 5.11 show the observed and fitted ozone partial pressure at six specific altitudes, so that the differences among SS-ANOVA fit, BSML fits and LAP fit can be seen more easily. The following code is used to generate these figures.

```
## Fitted values at the grid points for SS-ANOVA, BSML-C, BSML-S, LAP:
grid1 <- seq(11, 30, 0.1)
n.grid1 <- length(grid1)
grid2 <- seq(1, 1086, 5)
n.grid2 <- length(grid2)

X.grid <- expand.grid((grid1-11)/(30-11), (grid2-1)/1085)
x1.grid <- X.grid[,1]
x2.grid <- X.grid[,2]

fitted.gss <- matrix(predict(fit.gss, newdata=data.frame(x1=x1.grid,
              z2=3*x2.grid)), nrow=length(grid1), ncol=length(grid2))

baseslist.grid <- baseslist.gen(x1.grid, x2.grid, loc=unique(x1))
fitted.bsmlc <- matrix(predict(fit.bsmlc, bases.include=c(1,1,1,1,1),
                       new.baseslist=baseslist.grid, confint=F),
                       nrow=n.grid1, ncol=n.grid2)
fitted.bsmls <- matrix(predict(fit.bsmls, bases.include=c(1,1,1,1,1),
                       new.baseslist=baseslist.grid, confint=F),
                       nrow=n.grid1, ncol=n.grid2)
fitted.lap <- matrix(predict.LAP(fit2.lap, bases.include=c(1,1,1,1,1),
                     new.baseslist=baseslist.grid, confint=F)$fit,
                     nrow=n.grid1, ncol=n.grid2)

# Tick marks and labels for the plots:
month.tick2 <- c(1, 12, 25, 36, 49, 61, 73, 85, 98,
                 110, 122, 134, 146, 159, 171, 183, 195, 207)
month.labels2 <- c("9/06", "11/06", "1/07", "3/07", "5/07", "7/07",
                   "9/07", "11/07", "1/08", "3/08", "5/08", "7/08",
                   "9/08", "11/08", "1/09", "3/09", "5/09", "7/09")
alt.labels <- seq(11,30,1)
alt.tick <- seq(1, 191, 10)


# Figure 5.6:
levelplot(t(fitted.gss), aspect=0.5,
          ylab=list(label="Altitude (km)", cex=1.5),
```

```
          xlab=list(label="Month/Year", cex=1.5),
          main=list(label="SS-ANOVA fit of the ozone partial
                    pressure based on the training set", cex=2),
          col.regions = topo.colors(300),
          at=seq(min(fitted.gss, na.rm=TRUE),
                 max(ozone.train, na.rm=TRUE), length=300),
          colorkey=list(tick.number=10, labels=list(cex=1.2)),
          scales=list(x=list(cex=1,at=month.tick2, labels=month.labels2),
          y=list(cex=1.4, at=alt.tick, labels=alt.labels)))


# Figure 5.7:
levelplot(t(fitted.bsmlc), aspect=0.5,
          ylab=list(label="Altitude (km)", cex=1.5),
          xlab=list(label="Month/Year", cex=1.5),
          main=list(label="BSML-C fit of the ozone partial
                    pressure based on the training set", cex=2),
          col.regions = topo.colors(300),
          at=seq(min(fitted.bsmlc, na.rm=TRUE),
                 max(ozone.train, na.rm=TRUE), length=300),
          colorkey=list(tick.number=10, labels=list(cex=1.2)),
          scales=list(x=list(cex=1,at=month.tick2, labels=month.labels2),
          y=list(cex=1.4, at=alt.tick, labels=alt.labels)))


# Figure 5.8:
levelplot(t(fitted.bsmls), aspect=0.5,
          ylab=list(label="Altitude (km)", cex=1.5),
          xlab=list(label="Month/Year", cex=1.5),
          main=list(label="BSML-S fit of the ozone partial
                    pressure based on the training set", cex=2),
          col.regions = topo.colors(300),
          at=seq(min(fitted.bsmls, na.rm=TRUE),
                 max(ozone.train, na.rm=TRUE), length=300),
          colorkey=list(tick.number=10, labels=list(cex=1.2)),
          scales=list(x=list(cex=1,at=month.tick2, labels=month.labels2),
          y=list(cex=1.4, at=alt.tick, labels=alt.labels)))


# Figure 5.9:
levelplot(t(fitted.lap), aspect=0.5,
          ylab=list(label="Altitude (km)", cex=1.5),
```

```
          xlab=list(label="Month/Year", cex=1.5),
          main=list(label="LAP fit of the ozone partial
                    pressure based on the training set", cex=2),
          col.regions = topo.colors(300),
          at=seq(min(fitted.lap, na.rm=TRUE),
                 max(ozone.train, na.rm=TRUE), length=300),
          colorkey=list(tick.number=10, labels=list(cex=1.2)),
          scales=list(x=list(cex=1,at=month.tick2, labels=month.labels2),
          y=list(cex=1.4, at=alt.tick, labels=alt.labels)))


# Figure 5.10:
par(mfrow=c(3,1))
for(i in c(2,5,8)){
    plot(grid2, fitted.gss[10*(i-1)+1,], type="l",
         col=3, lwd=2, ylim=range(0,17),
         main=paste("Altitude = ", alt.train[i], "km"),
         cex.main=2, ylab="", xlab="", xaxt="n", yaxt="n")
    lines(grid2, fitted.lap[10*(i-1)+1,], col=2, lwd=2)
    lines(grid2, fitted.bsmlc[10*(i-1)+1,], col=1, lwd=2)
    points(date.uni, ozone.train[i,], col="gray60")
    axis(1,at=grid2[month.tick2], labels=month.labels2)
    axis(2, at=seq(0,17,2), labels=seq(0,17,2))
    mtext("Month/Year", side=1, line=3)
    mtext("Ozone Partial Pressure (mPa)", side=2, line=2.5)
}

# Figure 5.11:
par(mfrow=c(3,1))
for(i in c(11,14,17)){
    plot(grid2, fitted.gss[10*(i-1)+1,], type="l",
         col=3, lwd=2, ylim=range(0,17),
         main=paste("Altitude = ", alt.train[i], "km"),
         cex.main=2, ylab="", xlab="", xaxt="n", yaxt="n")
    lines(grid2, fitted.lap[10*(i-1)+1,], col=2, lwd=2)
    lines(grid2, fitted.bsmlc[10*(i-1)+1,], col=1, lwd=2)
    points(date.uni, ozone.train[i,], col="gray60")
    axis(1,at=grid2[month.tick2], labels=month.labels2)
    axis(2, at=seq(0,17,2), labels=seq(0,17,2))
    mtext("Month/Year", side=1, line=3)
    mtext("Ozone Partial Pressure (mPa)", side=2, line=2.5)
}
```

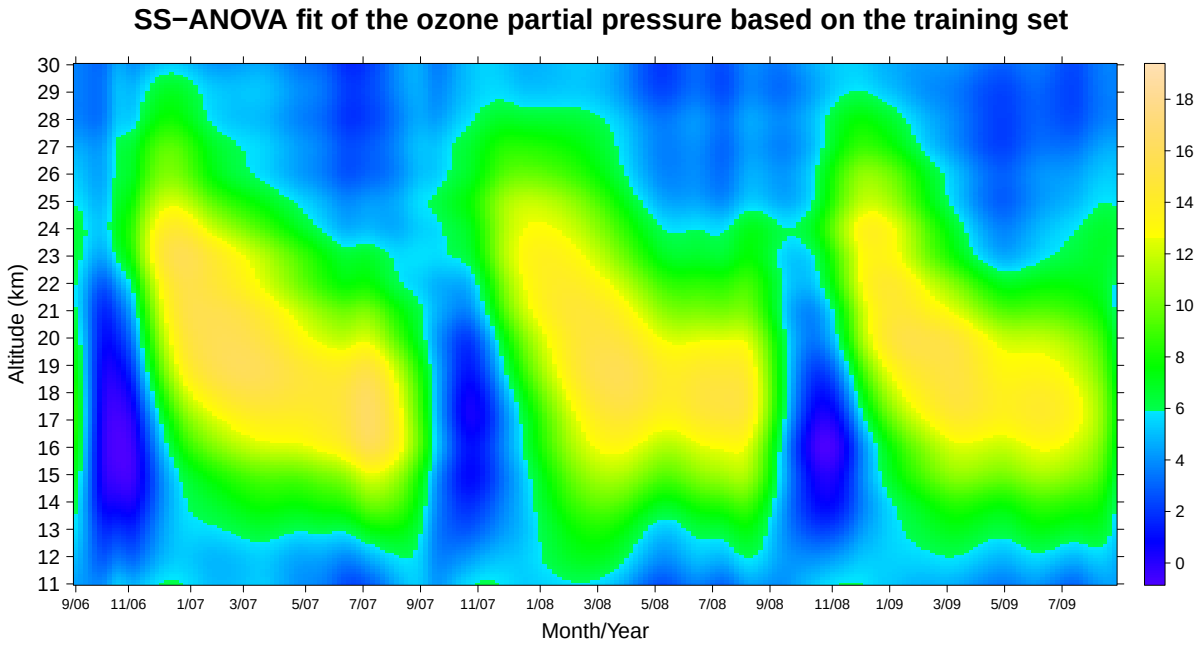


Figure 5.6: Level plot of SS-ANOVA fit of the ozone partial pressure based on the training set, for 191 different altitudes and 218 different dates.

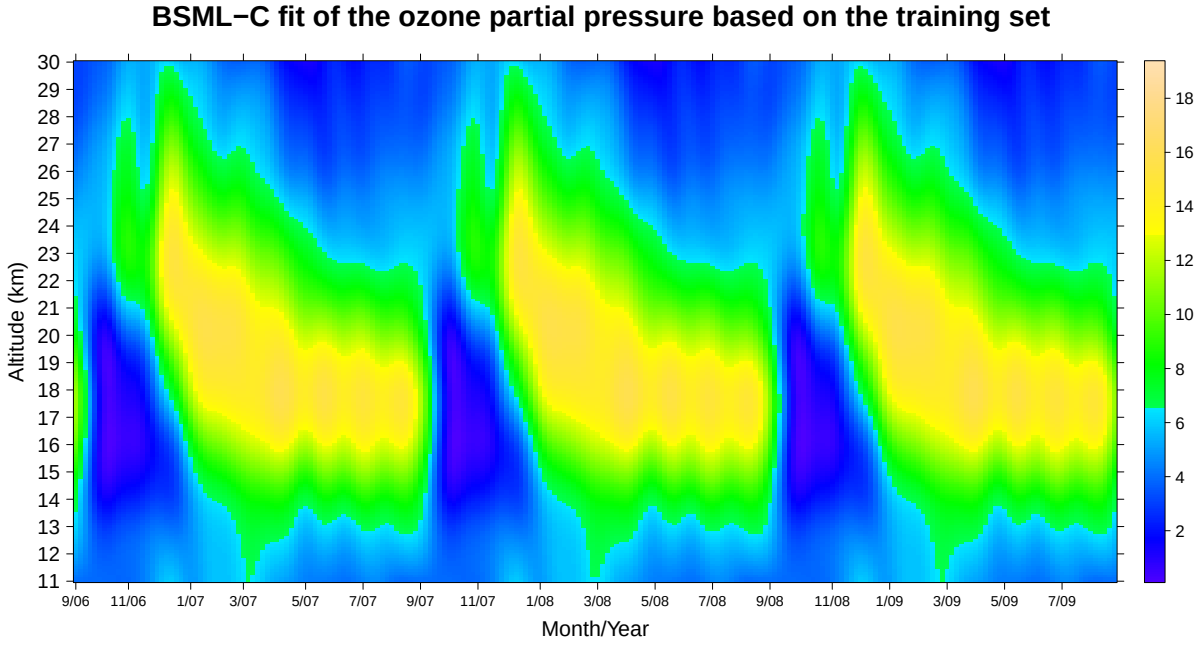


Figure 5.7: Level plot of the BSML-C fit of the ozone partial pressure based on the training set, for 191 different altitudes and 218 different dates.

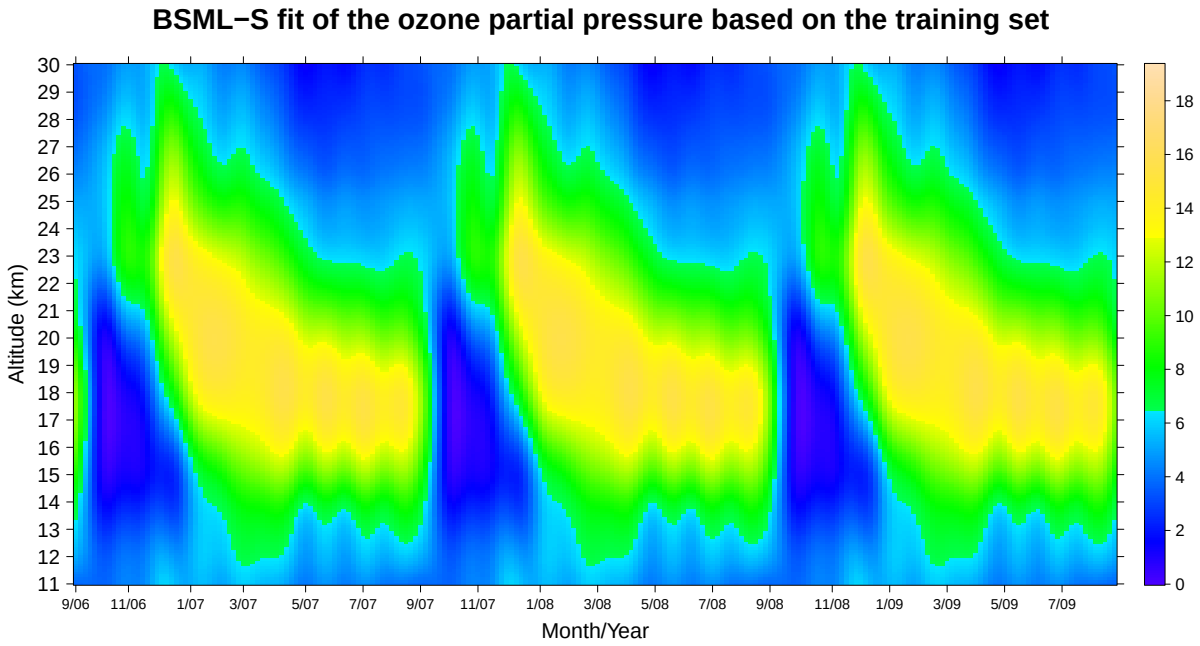


Figure 5.8: Level plot of the BSML-S fit of the ozone partial pressure based on the training set, for 191 different altitudes and 218 different dates.

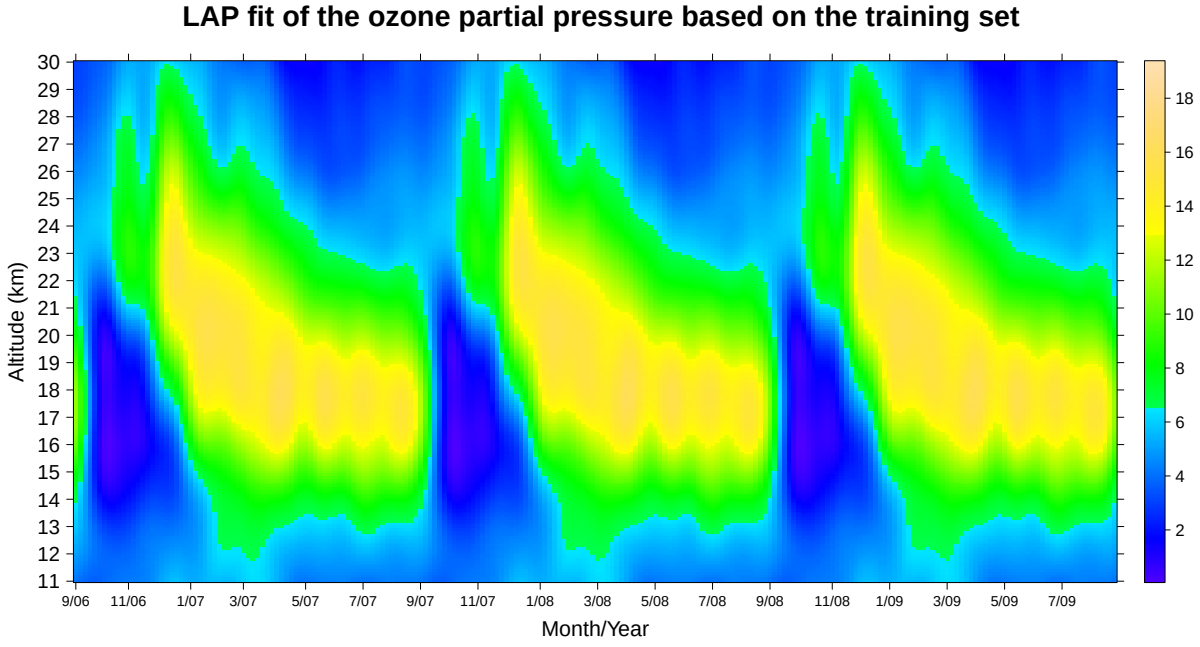


Figure 5.9: Level plot of the LAP fit of the ozone partial pressure based on the training set, for 191 different altitudes and 218 different dates.

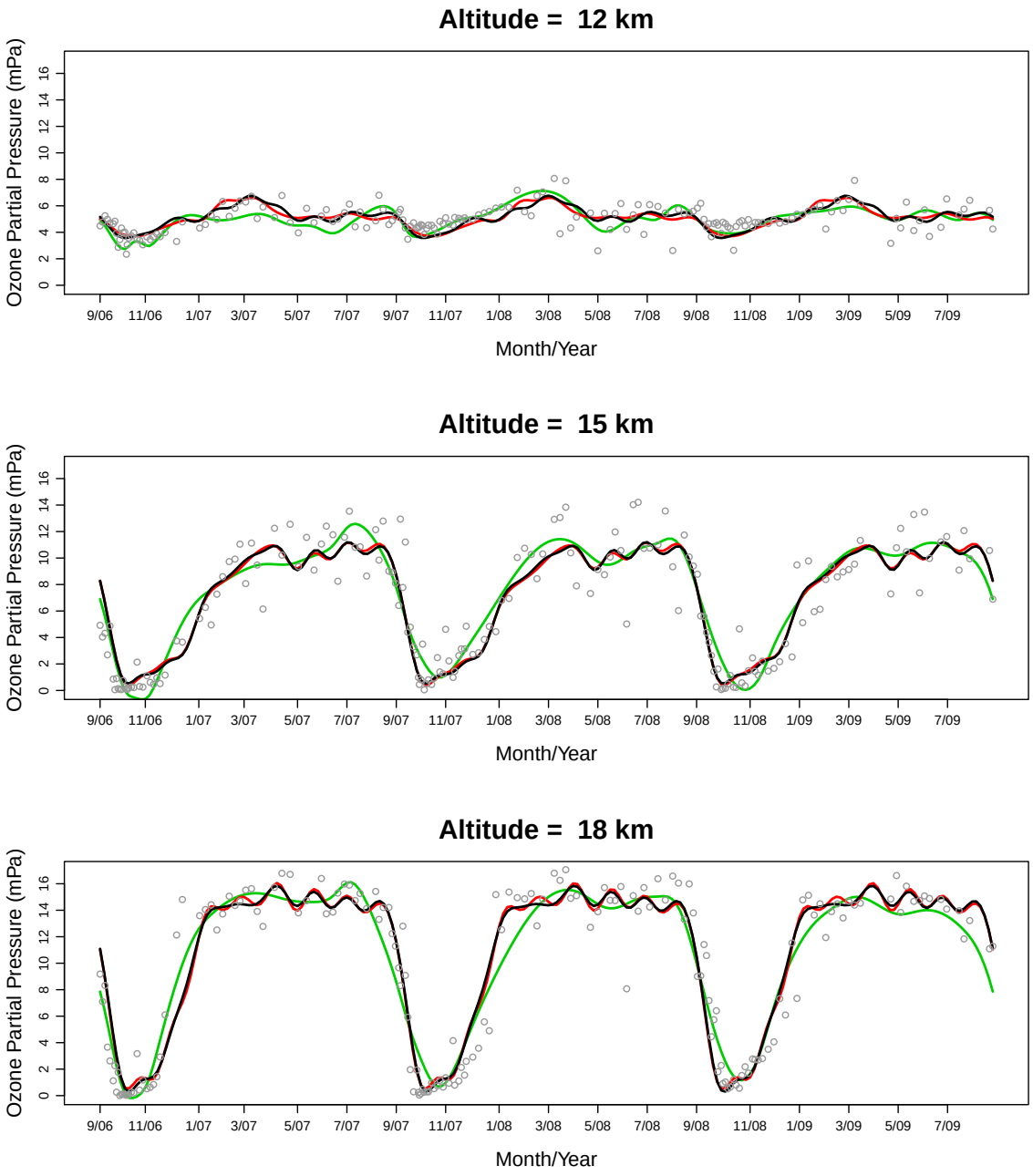


Figure 5.10: SS-ANOVA, BSML-C, BSML-S, and LAP fits of the ozone pressure at altitudes 12 km, 15 km, and 18 km. Here — represents the SS-ANOVA fit, — represents the LAP fit, — represents the BSML-C fit. The BSML-S fit is omitted because it is very close to the BSML-C fit. Circles represent the observations.

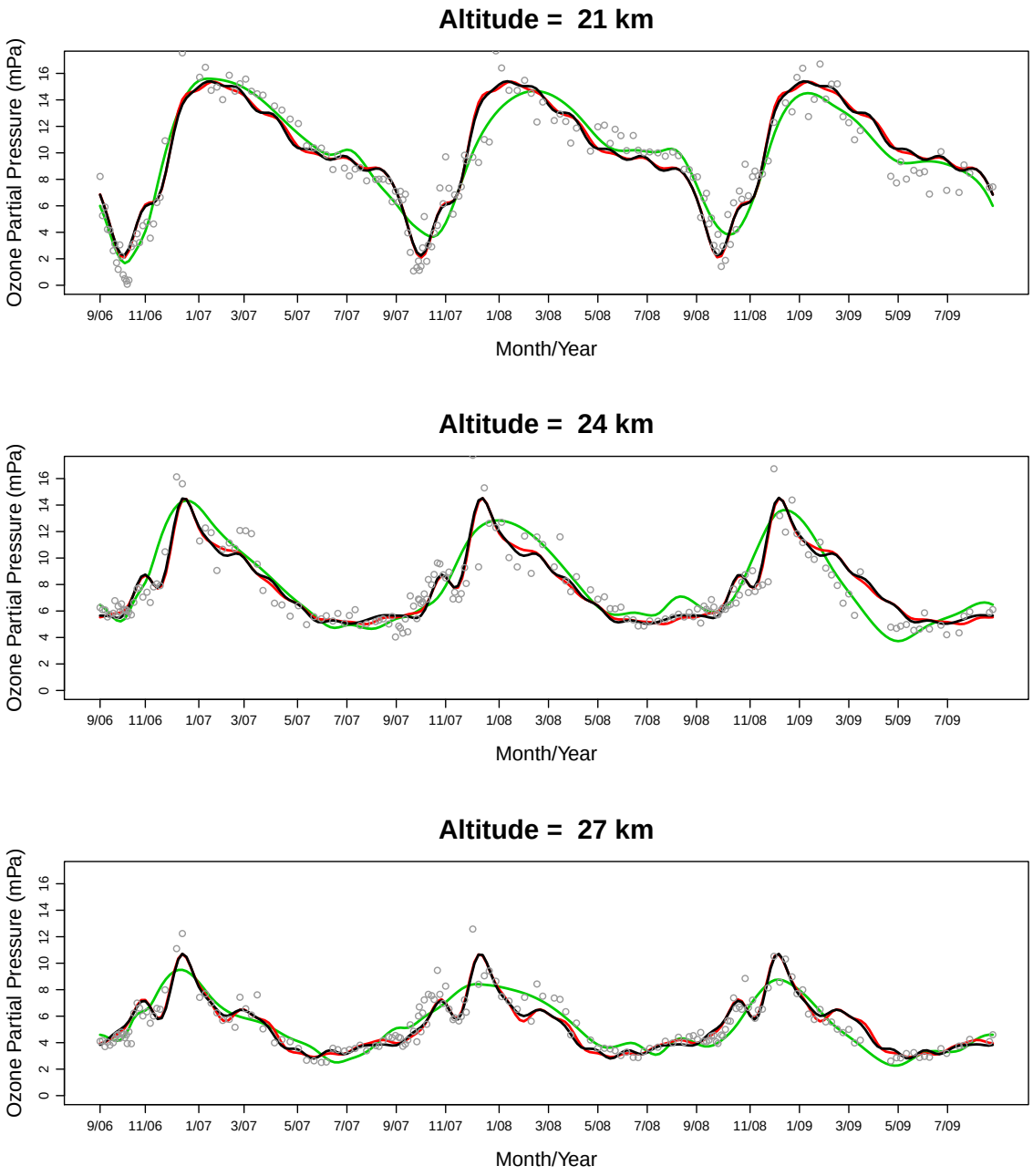


Figure 5.11: SS-ANOVA, BSML-C, BSML-S, and LAP fits of the ozone pressure at altitudes 21 km, 24 km, and 27 km. Here ___ represents the SS-ANOVA fit, ___ represents the LAP fit, ___ represents the BSML-C fit. The BSML-S fit is omitted because it is very close to the BSML-C fit. Circles represent the observations.

Based on the plots above, we can see that the BSML and LAP procedures have similar fits, whereas the SS-ANOVA model tends to oversmooth the data. Among all the 462 basis functions used in the BSML and LAP procedures, 86 of them are selected by LAP. In comparison, BSML-C and BSML-S have selected 67 and 72 bases respectively. The model returned from the LAP procedure has the lowest residual sum of squares compared to other three models. For fair comparison, we compute the predictive squared error (PSE) for the test set using the following code:

```
# PSE from the SS-ANOVA model:
pred.gss <- predict(fit.gss, newdata=data.frame(x1=x1.test,z2=3*x2.test))
( PSE.gss <- mean((pred.gss - ozone.test)^2, na.rm=TRUE) )

# PSE from the BSML-C procedure:
pred.bsmlc <- predict(fit.bsmlc, bases.include=c(1,1,1,1,1),
                      new.baseslist=baseslist.test, confint=F)
( PSE.bsmlc <- mean((pred.bsmlc - as.vector(ozone.test))^2, na.rm=TRUE) )

# PSE from the BSML-S procedure:
pred.bsmls <- predict(fit.bsmls, bases.include=c(1,1,1,1,1),
                      new.baseslist=baseslist.test, confint=F)
( PSE.bsmls <- mean((pred.bsmls - as.vector(ozone.test))^2, na.rm=TRUE) )

# PSE from the look-ahead procedure:
pred.lap <- predict.LAP(fit2.lap, bases.include=c(1,1,1,1,1),
                        new.baseslist=baseslist.test, confint=F)
( PSE.lap <- mean((pred.lap$fit - as.vector(ozone.test))^2, na.rm=TRUE) )
```

We found that `PSE.gss` = 2.191, `PSE.bsmlc` = 1.863, `PSE.bsmls` = 1.900, and `PSE.lap` = 1.826. Therefore, because the LAP procedure returns the lowest PSE, we conclude that LAP has the best performance compared to other three methods.

For the look-ahead procedure, here we construct the 95% bootstrap confidence intervals for the seasonal and nonseasonal effects at the same grid points of `altitude` and `day` used earlier. The seasonal effect includes the main effect of `day` and the interaction effect between `altitude` and `day`, which can be obtained by using the bases in $\mathcal{L}_2$ and $\mathcal{L}_3$ that

have been selected by LAP. We can estimate the seasonal effect alone by specifying the last two elements in the `bases.include` argument of the `predict.LAP` function to be 1 and others be 0, as shown in the code below. The nonseasonal effect includes the constant and the main effect of the latitude, which can be obtained by using the bases in $\mathcal{L}_0$ and $\mathcal{L}_1$ that have been selected by LAP. We can estimate the nonseasonal effect alone by specifying the first three elements in the `bases.include` argument of the `predict.LAP` function to be 1 and others be 0, where the first two elements in the `bases.include` corresponds to the null basis functions. For the same six altitudes studied earlier, the seasonal effects are plotted in Figure 5.12 and 5.13. The nonseasonal effect is shown in 5.14. Here is the `R` code we used to generate these figures.

```
# Seasonal fit and its confidence interval:
lap.season.fit <- predict.LAP(fit2.lap, bases.include=c(0,0,0,1,1),
new.baseslist=baseslist.grid, confint=T)
season.lap <- matrix(lap.season.fit$fit, nrow=n.grid1, ncol=n.grid2)
season.lap.lower <- matrix(lap.season.fit$lower,
                           nrow=n.grid1, ncol=n.grid2)
season.lap.upper <- matrix(lap.season.fit$upper,
                           nrow=n.grid1, ncol=n.grid2)

# Nonseasonal fit and its confidence interval:
lap.nonseason.fit <- predict.LAP(fit2.lap, bases.include=c(1,1,1,0,0),
new.baseslist=baseslist.grid, confint=T)
nonseason.lap <- matrix(lap.nonseason.fit$fit, nrow=n.grid1,ncol=n.grid2)
nonseason.lap.lower <- matrix(lap.nonseason.fit$lower,
                              nrow=n.grid1, ncol=n.grid2)
nonseason.lap.upper <- matrix(lap.nonseason.fit$upper,
                              nrow=n.grid1, ncol=n.grid2)

# Figure 5.12:
par(mfrow=c(3,1))
for(i in c(11, 41, 71)){
    plot(grid2, season.lap[i,], col=2, lwd=2,
         main=paste("Seasonal Effect \n Altitude = ", grid1[i], "km"),
         ylim=range(min(season.lap.lower[i,], na.rm=T),
                    max(season.lap.upper[i,], na.rm=T)),
         xlab="", ylab="", cex.main=1.5, type="l", xaxt="n")
```

```
    axis(1,at=grid2[month.tick2], labels=month.labels2)
    lines(grid2, season.lap.lower[i,], col=1, lwd=2, lty=2)
    lines(grid2, season.lap.upper[i,], col=1, lwd=2, lty=2)
    mtext("Month/Year", side=1, line=3)
    mtext("Ozone Partial Pressure (mPa)", side=2, line=2.5)
}

# Figure 5.13:
par(mfrow=c(3,1))
for(i in c(101,131,161)){
    plot(grid2, season.lap[i,], col=2, lwd=2,
         main=paste("Seasonal Effect \n Altitude = ", grid1[i], "km"),
         ylim=range(min(season.lap.lower[i,], na.rm=T),
                    max(season.lap.upper[i,], na.rm=T)),
         xlab="", ylab="", cex.main=1.5, type="l", xaxt="n")
    axis(1,at=grid2[month.tick2], labels=month.labels2)
    lines(grid2, season.lap.lower[i,], col=1, lwd=2, lty=2)
    lines(grid2, season.lap.upper[i,], col=1, lwd=2, lty=2)
    mtext("Month/Year", side=1, line=3)
    mtext("Ozone Partial Pressure (mPa)", side=2, line=2.5)
}

# Figure 5.14:
par(mfrow=c(1,1))
plot(nonseason.lap[,1], grid1, col=2, lwd=2, main="Nonseasonal Effect",
    xlim=range(min(nonseason.lap.lower[,1]-1, na.rm=T),
               max(nonseason.lap.upper[,1]+1, na.rm=T)),
    xlab="", ylab="", cex.main=1.5, type="l", xaxt="n", yaxt="n")
axis(1, at=2:12, labels=2:12)
axis(2, at=alt.train, labels=alt.train)
lines(nonseason.lap.lower[,1], grid1, col=1, lwd=2, lty=2)
lines(nonseason.lap.upper[,1], grid1, col=1, lwd=2, lty=2)
mtext("Altitude (km)", side=2, line=3)
mtext("Ozone Partial Pressure (mPa)", side=1, line=2.5)
```

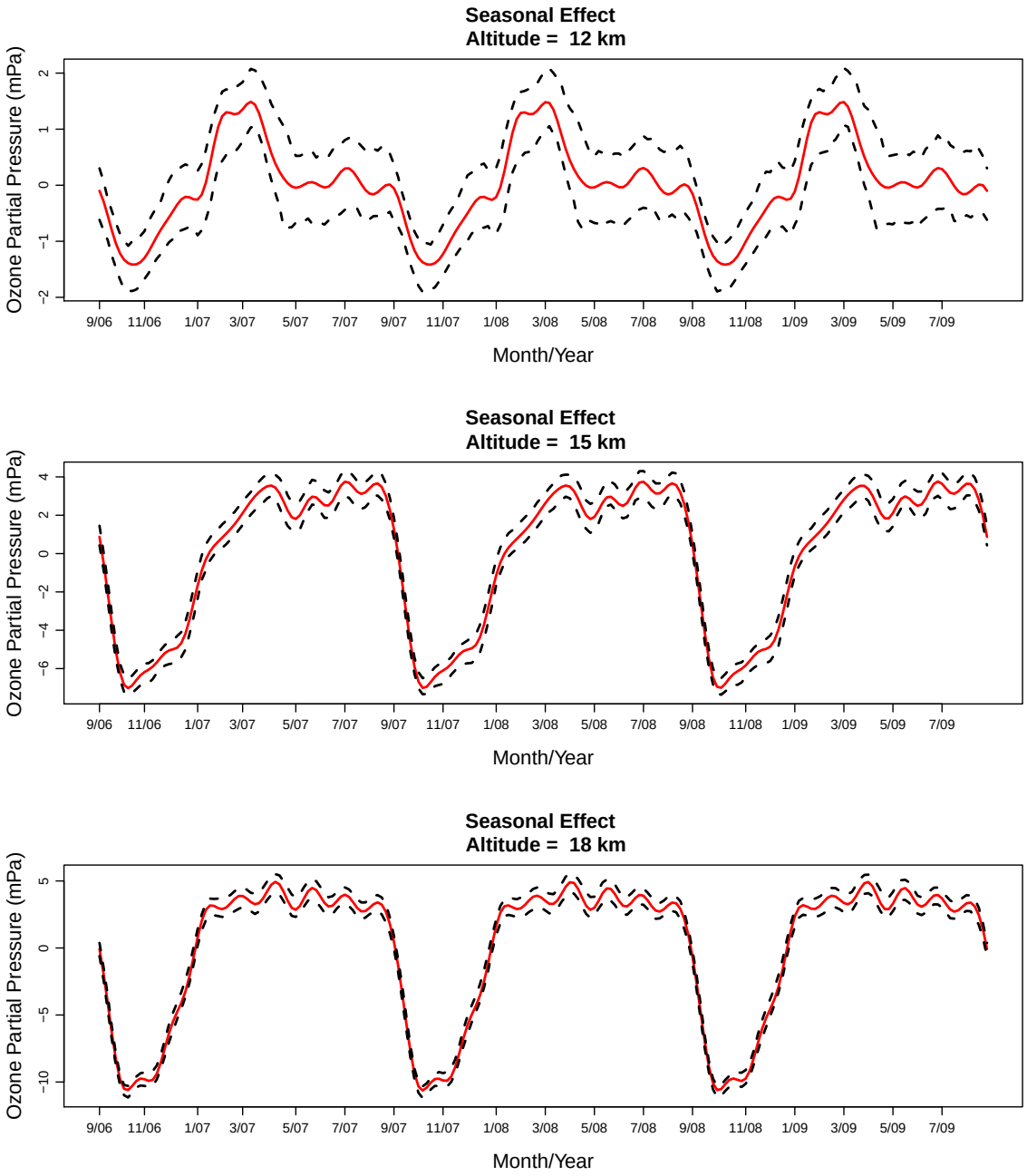


Figure 5.12: Seasonal effect at altitudes 12 km, 15 km, 18 km. The red solid line is the fit. The black dashed lines are the lower and upper bounds of the 95% bootstrap confidence interval.

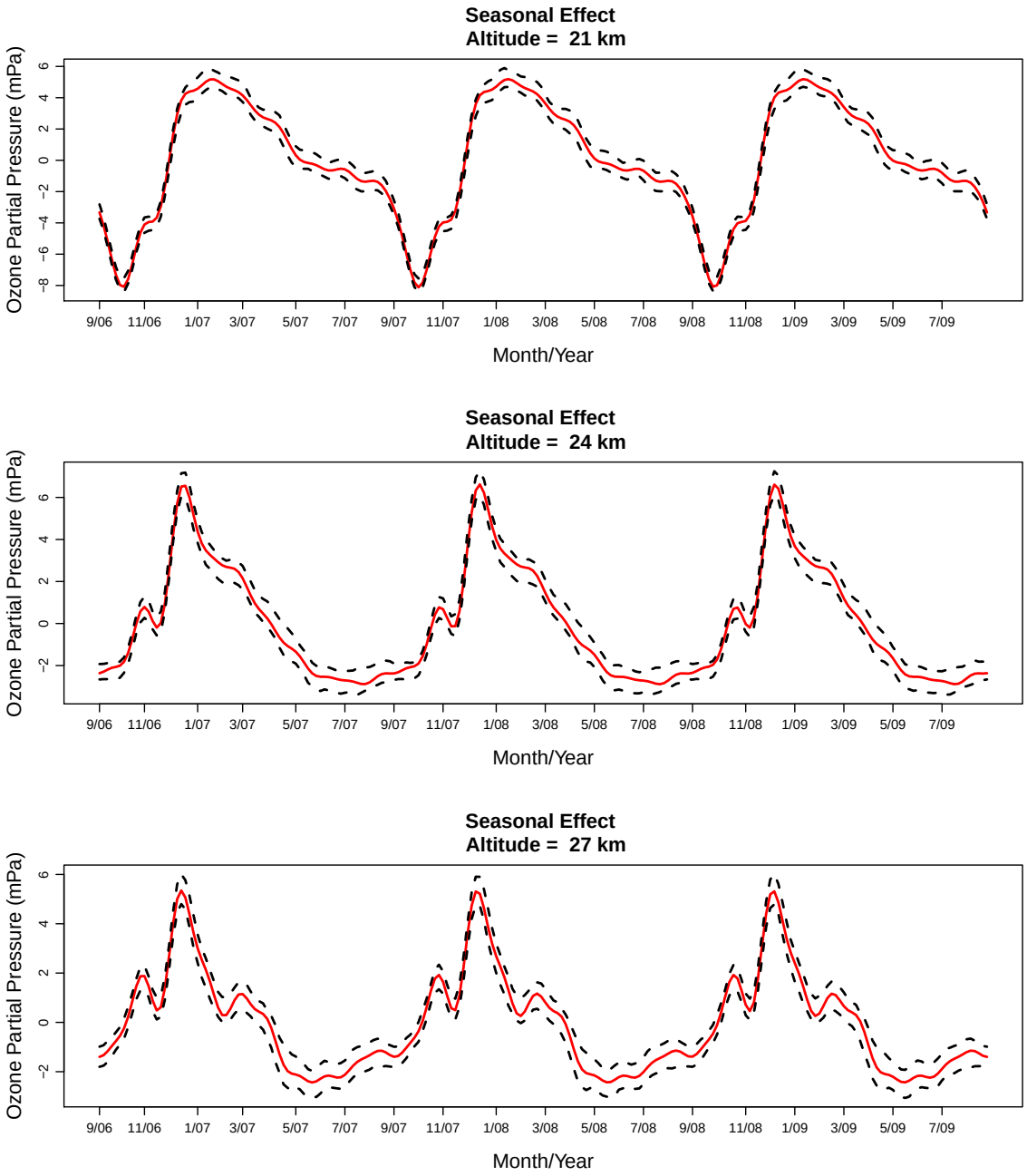


Figure 5.13: Seasonal effect at altitudes 21 km, 24 km, and 27 km. The red solid line is the fit. The black dashed lines are the lower and upper bounds of the 95% bootstrap confidence interval.

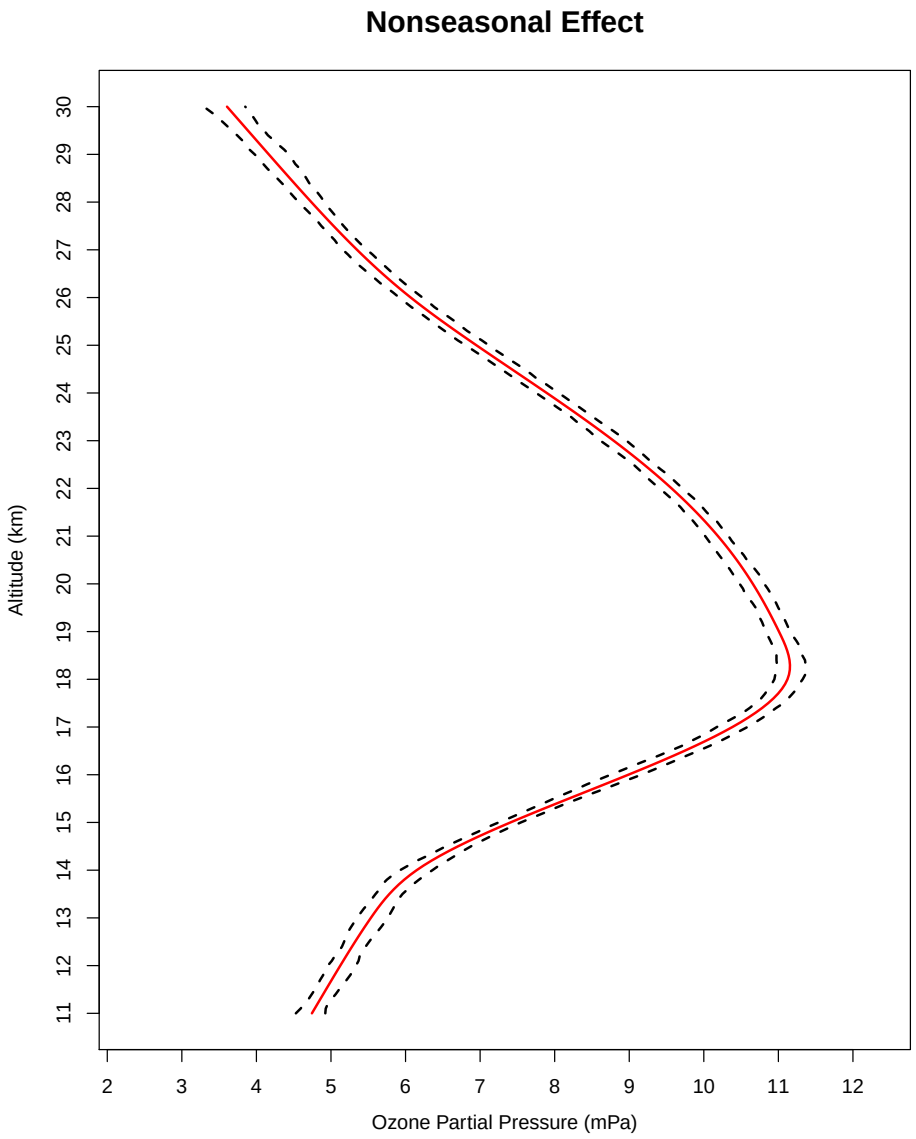


Figure 5.14: Nonseasonal effect. The red solid line is the fit. The black dashed lines are the lower and upper bounds of the 95% bootstrap confidence interval.

Based on Figures 5.12, 5.13, and 5.14, we can see that the seasonal effect is different at different altitudes. From Figure 5.12 we can observe the clear drop of ozone pressure during Spring time at low altitudes. However, this pattern no longer exists at high altitudes, as shown in the middle and bottom panels of Figure 5.13. The nonseasonal

effect does not depend on the dates, since it is fitted using only basis functions related to the variable `altitude`. Figure 5.14 suggests that in general, we should expect the ozone partial pressure to first increase and then decrease as the altitude increases. This finding agrees with the general pattern of ozone distribution in the stratosphere reported in literature. For example, Watson et al. (1986) mentioned that "the ozone layer has a continuous distribution with a peak concentration in the lower stratosphere between about 20 and 25 kilometers altitude".

Bibliography

H. Akaike. A new look at the statistical model identification. *IEEE Transactions on Automatic Control*, 19(6):716–723, 1974.

D. P. Bertsekas, J. N. Tsitsiklis, and C. Wu. Rollout algorithms for combinatorial optimization. *Journal of Heuristics*, 3(3):245–262, 1997. doi: http://dx.doi.org/10.1023/A:1009635226865.

S. Bjerve, K. A. Doksum, and B. S. Yandell. Uniform confidence bounds for regression based on a simple moving average. *Scandinavian Journal of Statistics*, 12(2):159–169, 1985.

G. Claeskens and I. V. Keilegom. Bootstrap confidence bands for regression curves and their derivatives. *The Annals of Statistics*, 31(6):1852–1884, 2003.

P. Craven and G. Wahba. Smoothing noisy data with spline functions. *Numerische Mathematik*, 31(4):377–403, 1978.

D. J. Cummins, T. G. Filloon, and D. Nychka. Confidence intervals for nonparametric curve estimates: toward more uniform pointwise coverage. *Journal of the American Statistical Association*, 96(453):233–246, 2001.

B. Efron. Nonparametric standard errors and confidence intervals. *Canadian Journal of Statistics*, 9(2):139–158, 1981. doi: http://dx.doi.org/10.2307/3314608.

B. Efron. *The Jackknife, the Bootstrap and Other Resampling Plans*, volume CBMS-NSF Regional Conference Series in Applied Mathematics, Vol. 38. SIAM, Philadelphia, 1982.

R. L. Eubank. *Spline Smoothing and Nonparametric Regression*. Marcel Dekker, second edition, 1999.

R. L. Eubank and P. L. Speckman. Confidence bands in nonparametric regression. *Journal of the American Statistical Association*, 88(424):1287–1301, 1993.

J. Fan and R. Li. Variable selection via nonconcave penalized likelihood and its oracle properties. *Journal of the American Statistical Association*, 96(456):1348–1360, 2001.

J. Friedman. Multivariate adaptive regression splines. *The Annals of Statistics*, 19(1): 1–67, 1991.

J. Friedman, T. Hastie, H. Höfling, and R. Tibshirani. Pathwise coordinate optimization. *The Annals of Applied Statistics*, 1(2):302–332, 2007. doi: http://dx.doi.org/10.1214/07-AOAS131.

J. Friedman, T. Hastie, and R. Tibshirani. *glmnet: Regularization Paths for Generalized Linear Models via Coordinate Descent*, 2010. URL `http://www.jstatsoft.org/v33/i01/`. R package version 2.0-5.

D. Frost and R. Dechter. Look-ahead value ordering for constraint satisfaction problems. *In Proceedings of the Fourteenth International Joint Conference on Artificial Intelligence*, pages 572–578, 1995.

T. Gasser, L. Sroka, and C. Jennen-Steinmetz. Residual variance and residual pattern in nonlinear regression. *Biometrika*, 73(3):625–633, 1986.

G. H. Golub and C. F. V. Loan. *Matrix Computations*. The Johns Hopkins University Press, fourth edition, 2013.

C. Gu. *gss: General Smoothing Splines*, 2013a. URL `http://CRAN.R-project.org/package=gss`. R package version 2.1-0.

C. Gu. *Smoothing Spline ANOVA Models*. Springer-Verlag New York, second edition, 2013b.

P. Hall. Effect of bias estimation on coverage accuracy of bootstrap confidence intervals for a probability density. *The Annals of Statistics*, 20(2):675–694, 1992a.

P. Hall. On bootstrap confidence intervals in nonparametric regression. *The Annals of Statistics*, 20(2):695–711, 1992b.

P. Hall and J. Horowitz. A simple bootstrap method for constructing nonparametric confidence bands for functions. *The Annals of Statistics*, 41(1):1892–1921, 2013.

W. Härdle and A. W. Bowman. Bootstrapping in nonparametric regression: local adaptive smoothing and confidence bands. *Journal of the American Statistical Association*, 83(401):102–110, 1988.

W. Härdle and J. S. Marron. Bootstrap simultaneous error bars for nonparametric regression. *The Annals of Statistics*, 19(2):778–796, 1991.

W. Härdle, S. Huet, and E. Jolivet. Better bootstrap confidence intervals for regression curve estimation. *Statistics*, 26(4):287–306, 1995.

T. Hastie and J. Qian. Glmnet vignette, 2014. URL `http://www.stanford.edu/~hastie/glmnet/glmnet_alpha.html`.

T. Hastie and R. Tibshirani. *Generalized Additive Models*. Chapman and Hall, 1990.

T. Hastie, R. Tibshirani, F. Leisch, K. Hornik, and B. D. Ripley. *mda: Mixture and flexible discriminant analysis*, 2015. URL `http://CRAN.R-project.org/package=mda`. S original by Trevor Hastie and Robert Tibshirani. Original R port by Friedrich Leisch and Kurt Hornik and Brian D. Ripley. R package version 0.4-8.

Y. Lin and H. H. Zhang. Component selection and smoothing in multivariate nonparametric regression. *The Annals of Statistics*, 34(5):2272–2297, 2006. doi: http://dx.doi.org/10.1214/009053606000000722.

Z. Luo and G. Wahba. Hybrid adaptive splines. *Journal of the American Statistical Association*, 92(437):107–116, 1997.

T. L. McMurry and D. N. Politis. Bootstrap confidence intervals in nonparametric regression with built-in bias correction. *Statistics & Probability Letters*, 78(15):2463–2469, 2008.

M. H. Neumann. Automatic bandwidth choice and confidence intervals in nonparametric regression. *The Annals of Statistics*, 23(6):1937–1959, 1995.

M. H. Neumann and J. Polzehl. Simultaneous bootstrap confidence bands in nonparametric regression. *Journal of Nonparametric Statistics*, 9(4):307–333, 1998.

D. Picard and K. Tribouley. Adaptive confidence interval for pointwise curve estimation. *The Annals of Statistics*, 28(1):298–335, 2000.

W. Qian and Y. Yang. Model selection via standard error adjusted adaptive lasso. *Annals of the Institute of Statistical Mathematics*, 65(2):295–318, 2013.

P. Qiu. *Image Processing and Jump Regression Analysis*. John Wiley & Sons Publication, 2005.

J. Rice. Bandwidth choice for nonparametric regression. *The Annals of Statistics*, 12(4):1215–1230, 1984.

J. Ruanaidh and W. Fitzgerald. *Numerical Bayesian Methods Applied to Signal Processing*. Springer-Verlag New York, 1996.

G. Schwarz. Estimating the dimension of a model. *The Annals of Statistics*, 6(2):461–464, 1978.

G. A. Seber and A. J. Lee. *Linear Regression Analysis*. John Wiley & Sons Publication, second edition, 2003.

J. C. Sklar, J. Wu, W. Meiring, and Y. Wang. Non-parametric regression with basis selection from multiple libraries. *Technometrics*, 55(2):189–201, 2013. doi: http://dx.doi.org/10.1080/00401706.2012.739104.

S. Solomon. Stratospheric ozone depletion: a review of concepts and history. *Reviews of Geophysics*, 37(3):275–316, 1999.

J. Sun and C. R. Loader. Simultaneous confidence bands for linear regression and smoothing. *The Annals of Statistics*, 22(3):1328–1345, 1994.

T. Tong and Y. Wang. Estimating residual variance in nonparametric regression using least squares. *Biometrika*, 92(4):821–830, 2005.

S. Voss, A. Fink, and C. Duin. Looking ahead with the pilot method. *Annals of Operations Research*, 136(1):285–302, 2005. doi: http://dx.doi.org/10.1007/s10479-005-2060-2.

G. Wahba. Bayesian confidence intervals for the cross-validated smoothing spline. *Journal of the Royal Statistical Society, Series B*, 45(1):133–150, 1983.

G. Wahba. *Spline models for observational data*, volume CBMS-NSF Regional Conference Series in Applied Mathematics, Vol. 59. SIAM, Philadelphia, 1990.

Y. Wang. *Smoothing Splines Methods and Applications*. CRC Press, 2011.

Y. Wang and C. Ke. *assist: A Suite of S-Plus Functions Implementing Smoothing Splines*, 2015. URL `http://CRAN.R-project.org/package=assist`. R package version 3.1.3.

Y. Wang and G. Wahba. Bootstrap confidence intervals for smoothing splines and their comparison to bayesian confidence intervals. *Journal of Statistical Computation and Simulation*, 51(2-4):263–279, 1995.

R. T. Watson, M. A. Geller, R. S. Stolarski, and R. F. Hampson. Present state of knowledge of the upper atmosphere: An assessment report. *NASA Reference Publication 1162*, 1986.

J. Wu, J. Sklar, Y. Wang, and W. Meiring. *bsml: Basis Selection from Multiple Libraries*, 2012. URL `https://cran.r-project.org/src/contrib/Archive/bsml/`. R package version 1.5-1.

Y. Xia. Bias-corrected confidence bands in nonparametric regression. *Journal of the Royal Statistical Society. Series B. Statistical Methodology*, 60(4):797–811, 1998.

J. Ye. On measuring and correcting the effects of data mining and model selection. *Journal of the American Statistical Association*, 93(441):120–131, 1998.

H. H. Zhang and C.-Y. Lin. *cosso: Fit Regularized Nonparametric Regression Models Using COSSO Penalty*, 2013. URL `http://CRAN.R-project.org/package=cosso`. R package version 2.1-1.

J. L. Zhang and J. S. Liu. A new sequential importance sampling method and its application to the two-dimensional hydrophobichydrophilic model. *Journal of Chemical Physics*, 117(7):3492–3498, 2002.

J. L. Zhang, M. T. Lin, J. S. Liu, and R. Chen. Lookahead and piloting strategies for variable selection. *Statistica Sinica*, 17(3):985–1003, 2007.

H. Zou. The adaptive lasso and its oracle properties. *Journal of the American Statistical Association*, 101(476):1418–1429, 2006.